LOVELY PROFESSIONAL UNIVERSITY

MODERN WEB PROGRAMMING TOOLS AND TECHNIQUES – I

Edited By
Kumar Vishal

CONTENT

Unit 1: The .Net Framework 1
Kumar Vishal, Lovely Professional University

Unit 2: Visual Studio 21
Kumar Vishal, Lovely Professional University

Unit 3: Web Form Fundamentals 57
Kumar Vishal, Lovely Professional University

Unit 4: Web Controls 82
Kumar Vishal, Lovely Professional University

Unit 5: State Management 115
Kumar Vishal, Lovely Professional University

Unit 6: Error Handling Logging and Tracing 146
Kumar Vishal, Lovely Professional University

Unit 7: Validation 183
Kumar Vishal, Lovely Professional University

Unit 8; Rich Controls 203
Kumar Vishal, Lovely Professional University

Unit 9: User Controls and Graphics 228
Sarabjit Kumar, Lovely Professional University

Unit 10: Themes and Styles 252
Sarabjit Kumar,Lovely Professional University

Unit 11: Master Page Basics 277
Sarabjit Kumar, Lovely Professional University

Unit 12: ADO.NET and Data Binding 299
Sarabjit Kumar, Lovely Professional University

Unit 13: Website Security 344
Sarabjit Kumar,Lovely Professional University

Unit 14: Deploying Website 367
Sarabjit Kumar, Lovely Professional University

SYLLABUS

Modern Web Programming Tools and Techniques – I

Objectives: This course provides the deep coverage of web development tools and technologies

Sr. No.	Topics
1.	**The .Net Framework:** HTML and HTML forms, Server side programming, client programming, Common language runtime, .Net Class library
2.	**Visual Studio**: Creating Websites, Designing a webpage, The anatomy of a Web Form, Writing Code, Visual Studio Debugging.
3.	**Web Form Fundamentals:** ASP.NET application, Introducing Server Controls, HTML control classes, The Page Class, Application events, ASP.NET configuration.
4.	**Web Controls:** Web Control Classes, List Controls, Web Control Events and AutoPostBack, A simple Web Page
5.	**State Management:** View State, Transferring Information Between Pages, Cookies, Session State, Session State Configuration, Application State.
6.	**Error Handling Logging and Tracing:** Common errors, Handling Exceptions, Throwing your own Exceptions, Logging Exceptions, Error Pages, Page Tracing.
7.	**Validation:** Understanding Validation, The validation controls **Rich Controls:** The calendar, AdRotator, Pages with Multiple view, **User Controls and Graphics:** User Controls, Dynamic Graphics
8.	**Styles, Themes and Master Pages:** Styles, Themes, Master Page Basics.
9.	**ADO.NET and Data Binding:** Configuring your Database, ADO.NET basics, Direct Data Access. Single Value data binding, Repeated Value Data Binding.
10.	**Website Security:** ASP.NET security Model, Forms Authentication, Windows Authentication. **Deploying Web Site:** How to deploy the web site. On local IIS or remote IIS.

Unit 1: The .Net Framework

Notes

CONTENTS

Objectives

Introduction

1.1 HTML

1.2 Server-side Programming

1.2.1 Advantages of Server-side Programs

1.2.2 Types of Server-side Programs

1.3 Client-side Programming

1.4 Common Language Runtime

1.5 .Net Class Library

1.6 Role of .Net Class Libraries

1.7 Summary

1.8 Keywords

1.9 Self Assessment

1.10 Review Questions

1.11 Further Readings

Objectives

After studying this unit, you will be able to:

- Define know HTML and HTML forms
- Describe server side programming
- Describe client side programming
- Explain common language runtime
- Know .Net class library

Introduction

You would by now have been introduced to the Internet and the World Wide Web (often just called the Web) and how it has changed our lives. Today we have access to a wide variety of information through websites on the Internet. We can access a website if we have a connection to the Internet and a browser on our computer. Popular browsers are Microsoft Internet Explorer, Netscape Navigator and Opera. When you connect to a Website, your browser is presented with a file in a special format by the Web server on the remote computer. The contents of the file are stored in a special format using Hyper Text Markup Language, often called HTML This format is rendered, or interpreted, by the browser and you then see the page of the website from your computer.

Notes

HTML is one language in a class of markup languages, the most general form of which is Standard Generalized Markup Language, or SGML. Since SGML is complex, HTML was invented as a simple way of creating web pages that could be easily accessed by browsers. HTML is a special case of SGML.

HTML consists of tags and data. The tags serve to define what kind of data follows them, thereby enabling the browser to render the data in the appropriate form for the user to see. There are many tags in HTML, of which the few most important ones are introduced in this unit. HTML files usually have the extension ".htm" or ".html".

If you want to create Web pages, you need a tool to write the HTML code for the page. This can be a simple text editor if you are hand-coding HTML. You also have sophisticated HTML editors available that automate many (though not all) of the tasks of coding HTML. You also need a browser to be able to render your code so that you can see the results.

1.1 HTML

HTML stands for HyperText Markup Language. HTML provides a way of displaying Web pages with text and images or multimedia content. HTML is not a programming language, but a markup language. An HTML file is a text file containing small markup tags. The markup tags tell the Web browser, such as Internet Explorer or Netscape Navigator, how to display the page. An HTML file must have an htm or html file extension. These files are stored on the web server. So if you want to see the web page of a company, you should enter the URL (Uniform Resource Locator), which is the website address of the company in the address bar of the browser. This sends a request to the web server, which in turn responds by returning the desired web page. The browser then renders the web page and you see it on your computer.

HTML allows Web page publishers to create complex pages of text and images that can be viewed by anyone on the Web, regardless of what kind of computer or browser is being used. Despite what you might have heard, you don't need any special software to create an HTML page; all you need is a word processor (such as Microsoft Word) and a working knowledge of HTML. Fortunately, the basics of HTML are easy to master. However, you can greatly relieve tedium and improve your productivity by using a good tool. A simple tool is Microsoft FrontPage that reduces the need to remember and type in HTML tags. Still, there can always be situations where you are forced to handcode certain parts of the web page.

HTML is just a series of tags that are integrated into a document that can have text, images or multimedia content. HTML tags are usually English words (such as blockquote) or abbreviations (such as p for paragraph), but they are distinguished from the regular text because they are placed in small angle brackets. So the paragraph tag is <p>, and the blockquote tag is <blockquote>. Some tags dictate how the page will be formatted (for instance, <p> begins a new paragraph), and others dictate how the words appear (<b> makes text bold). Still others provide information - such as the title - that doesn't appear on the page itself. The first thing to remember about tags is that they travel in pairs. Most of the time that you use a tag - say <blockquote> - you must also close it with another tag - in this case, </blockquote>. Note the slash - / - before the word "blockquote"; that is what distinguishes a closing tag from an opening tag.

The basic HTML page begins with the tag <html> and ends with </html>. In between, the file has two sections - the header and the body.

The header - enclosed by the <head> and </head> tags - contains information about a page that will not appear on the page itself, such as the title. The body - enclosed by <body> and </body> - is where the action is. Everything that appears on the page is contained within these tags.

Notes

What is HTML?

H-T-M-L are initials that stand for HyperText Markup Language (computer people love initials and acronyms -- you'll be talking acronyms ASAP). Let me break it down for you:

Hyper is the opposite of linear. It used to be that computer programs had to move in a linear fashion. This before this, this before this, and so on. HTML does not hold to that pattern and allows the person viewing the World Wide Web page to go anywhere, any time they want.

Text is what you will use. Real, honest to goodness English letters.

Mark up is what you will do. You will write in plain English and then mark up what you wrote.

Language because they needed something that started with "L" to finish HTML and Hypertext Markup Louie didn't flow correctly. Because it's a language, really -- but the language is plain English.

HTML pages are of two types:

1. Static pages
2. Dynamic pages

Static Pages

Static pages, as the name indicates, comprise static content (text or images). So you can only see the contents of a web page without being able to have any interaction with it.

Dynamic Pages

Dynamic pages are those where the content of the web page depend on user input. So interaction with the user is required in order to display the web page. For example, consider a web page which requires a number to be entered from the user in order to find out if it is even or odd. When the user enters the number and clicks on the appropriate button, the number is sent to the web server, which in turn returns the result to the user in an HTML page.

It would be difficult to describe early websites as web applications. Instead, the first generation of websites often looked more like brochures, consisting mostly of fixed HTML pages that needed to be updated by hand.

A basic HTML page is a little like a word-processing document it contains formatted content that can be displayed on your computer, but it doesn’t actually do anything. The following example shows HTML at its simplest, with a document that contains a heading and single line of text:

```
<html>
       <head>
              <title>Sample Web Page</title>
       </head>
      <body>
           <h1>Sample Web Page Heading</h1>
           <p>This is a sample web page.</p>
      </body>
</html>
```

An HTML document has two types of content: the text and the elements (or tags) that tell the browser how to format it. The elements are easily recognizable, because they are designated with angled brackets (< >). HTML defines elements for different levels of headings, paragraphs,

Notes

hyperlinks, italic and bold formatting, horizontal lines, and so on. For example, <h1>Some Text</h1> uses the <h1> element. This element tells the browser to display Some Text in the Heading 1 style, which uses a large, bold font. Similarly, This is a sample web page. creates a paragraph with one line of text. The <head> element groups the header information together, including the title that appears in the browser window, while the <body> element groups together the actual document content that's displayed in the browser window.

Figure 1.1 shows this simple HTML page in a browser. Right now, this is just a fixed file (named sample_web_page_heading.htm) that contains HTML content. It has no interactivity, doesn't require a web server, and certainly can't be considered a web application.

Figure 1.1: Ordinary HTML: the "brochure" Site

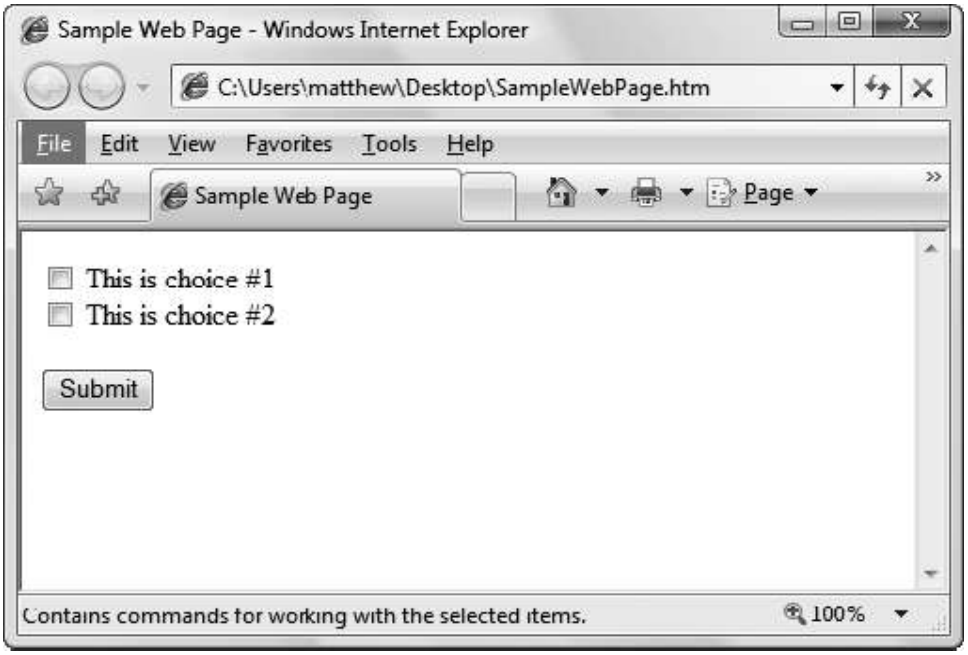
HTML 2.0 introduced the first seed of web programming with a technology called HTML forms. HTML forms expand HTML so that it includes not only formatting tags but also tags for graphical widgets, or controls. These controls include common ingredients such as drop-down lists, text boxes, and buttons. Here's a sample web page created with HTML form controls:

```
<html>
<head>
<title>Sample Web Page</title>
</head>
<body>
<form>
<input type="checkbox" />
This is choice #1<br />
<input type="checkbox" />
This is choice #2<br /><br />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Notes

In an HTML form, all controls are placed between the <form> and </form> tags. The preceding example includes two check boxes (represented by the ☐ element) and a button (represented by the <input type="submit" /> element). The <br/> element adds a line break in between lines. In a browser, this page looks like Figure 1.2.

Figure 1.2: An HTML Form



HTML forms allow web application developers to design standard input pages. When the user clicks the Submit button on the page shown in Figure 1.2, all the data in the input controls (in this case, the two check boxes) is patched together into one long string of text and sent to the web server. On the server-side, a custom application receives and processes the data.

Amazingly enough, the controls that were created for HTML forms more than ten years ago are still the basic foundation that you'll use to build dynamic ASP.NET pages! The difference is the type of application that runs on the server-side. In the past, when the user clicked a button on a form page, the information might have been e-mailed to a set account or sent to an application on the server that used the challenging Common Gateway Interface (CGI) standard. Today, you'll work with the much more capable and elegant ASP.NET platform.

HTML Tags

A tag is a reference in an HTML document which describes the style and structure of the document. All tags start with < and end with >. Tags which mark a beginning have no /. Tags which mark an ending have a / immediately after <, as in </.

```
<HTML>
<HEAD>
<TITLE>My Home Page</TITLE>
</HEAD>
<!-- Written by me -->
```

Notes

<!-- Created: yesterday -->

<!-- Last modified: today -->

<BODY>

This is where the text goes.

</BODY>

</HTML>

There are two kinds of tags. Those tags which require an ending, are called range tags. They are called "range tags" because they cover a range of text. Examples of range tags are <HTML>, <HEAD>, <BODY>, <TITLE>. Range tags require ending tags (</HTML>, </HEAD>, </BODY>, </TITLE>)

<HTML>

<HEAD>

<TITLE>My Home Page</TITLE>

</HEAD>

<!-- Written by me -->

<!-- Created: yesterday -->

<!-- Last modified: today -->

<BODY>

This is where the text goes.

</BODY>

</HTML>

Tag Attributes:

A tag is also sometimes called an element. You may think of HTML elements as nouns or verbs within a language. Nouns have adjectives and verbs have adverbs.

I have a cat.

What kind of a cat?

A big cat.

What color is your big cat?

I have a big black cat.

HEAD and BODY Tags

HTML documents are separated into two sections, the head and the body. The beginning of the head section is marked by the <HEAD> tag, and the end of the head section with the </HEAD> tag. Likewise, the beginning of the body section is marked by the <BODY> tag, and the end of the body section with the </BODY> tag. The example below highlights these tags.

<HTML>

<HEAD>

Notes

```
<TITLE>My Home Page</TITLE>
</HEAD>
<!-- Written by me -->
<!-- Created:  yesterday -->
<!-- Last modified:  today -->
<BODY>
This is where the text goes.
</BODY>
</HTML>
```

The TITLE Tag

The title of the page must be placed within the head section. The beginning of the title is marked by the <TITLE> tag, and the end with the </TITLE> tag. Most browsers display the title on the top portion of the frame of the page. Some browsers give the user the option to hide the display of the title. Many browsers also display the title of the document in the document title field, and use the title as the bookmark listing.

Titles should be short, but descriptive, and limited to one line, so they can fit within the frame of the page. You should avoid accentuation within the title because it is the window manager, and not the browser, that interprets the title. Some window managers do not support accents.

```
<HTML>
<HEAD>
<TITLE>My Home Page</TITLE>
</HEAD>
<!-- Written by me -->
<!-- Created:  yesterday -->
<!-- Last modified:  today -->
<BODY>
This is where the text goes.
</BODY>
</HTML>
```

The Comment Tag

The comment tag is different from the other tags for the following reasons. It can be placed anywhere in the HTML file; inside the head section, inside the body section, or in between the head and the body. It does not end with a /. It starts with an <!-- and ends with an -->.

It is always a good idea to put comments into your source code. In this simple example the comments state who is the author, the creation date of the document and the date of most recent update.

```
<HTML>
<HEAD>
```

Notes

```
<TITLE>My Home Page</TITLE>
</HEAD>
<!-- Written by me -->
<!-- Created:  yesterday -->
<!-- Last modified:  today -->
<BODY>
This is where the text goes.
</BODY>
</HTML>
```

The Paragraph Tag

To separate your text into paragraphs use the <P> tag. For example, the following HTML source code:

This is should be the end of my first paragraph in HTML.

This should be the start of my second paragraph in HTML.<P>

And this is should be my third paragraph in HTML.<P>

Would display this on the user's screen:

This is should be the end of my first paragraph in HTML. This should be the start of my second paragraph in HTML.

And this is should be my third paragraph in HTML.

Normally all text in an HTML document is treated like one long paragraph. Indented or blank lines which normally indicate start of paragraphs in text are ignored. The <P> tag is used to separate paragraphs of text within an HTML document. The <P> tag forces an end of line and forces a blank line before the next paragraph.

The <P> tag is only required to separate paragraphs in a document. It is not required after other tags which imply a line break.

The BR Tag

Normally different browsers decide where is the best place to line breaks in your HTML document. However, there may be times when you want to control where one line ends and the next begins. Use the <BR> tag to do this.

The <BR> tag is very similar to the <P> tag in that both separate text, but the <BR> tag does not include a blank line before starting the next line of text. In other words, the lines are single spaced, not double spaced. For example, the following HTML source code:

This is should be the end of my first paragraph in HTML.

This should be the start of my second paragraph in HTML.

And this is should be my third paragraph in HTML.

Would display this on the user's screen:

This is should be the end of my first paragraph in HTML. This should be the start of my second paragraph in HTML.

Notes

And this is should be my third paragraph in HTML.

The <BR> tag is only required when you want to control where the line breaks. It is not required after other tags which imply a line break, such as the horizontal rule tag.

The HR Tag

The <HR> tag draws a horizontal line across the page, like the line above this paragraph. It also acts as a paragraph break. There is no need to use the <BR> before or after the <HR> tag. For example, the following HTML source code:

A paragraph of text. <HR> Another paragraph of text.

Would display this on the user's screen:

A paragraph of text.

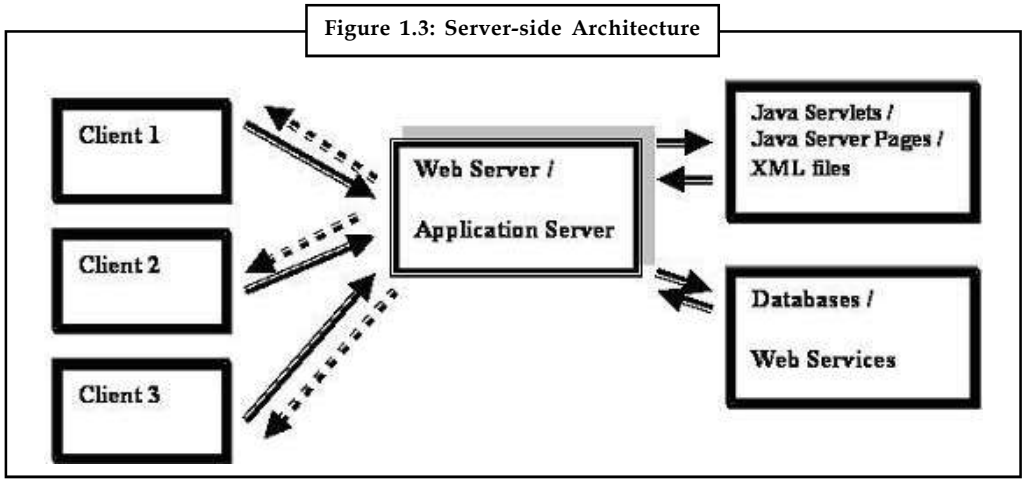
Another paragraph of text.

1.2 Server-side Programming

Though it is technically feasible to implement almost any business logic using client-side programs, logically or functionally it carries no ground when it comes to enterprise applications (e.g. banking, air ticketing, e-shopping etc.). To further explain, going by the client-side programming logic; a bank having 10,000 customers would mean that each customer should have a copy of the program(s) in his or her PC which translates to 10,000 programs! In addition, there are issues like security, resource pooling, concurrent access and manipulations to the database which simply cannot be handled by client-side programs. The answer to most of the issues cited above is – "Server-side Programming". Figure 1.3 illustrates Server-side architecture in the simplest terms.

To understand why ASP.NET was created, it helps to understand the problems of early web development technologies. With the original CGI standard, for example, the web server must launch a completely separate instance of the application for each web request. If the website is popular, the web server struggles under the weight of hundreds of separate copies of the application, eventually becoming a victim of its own success. Furthermore, technologies such as CGI provide a bare-bones programming environment. If you want higher-level features, like the ability to authenticate users, store personalized information, or display records you've retrieved from a database, you need to write pages of code from scratch. Building a web application this way is tedious and error-prone.

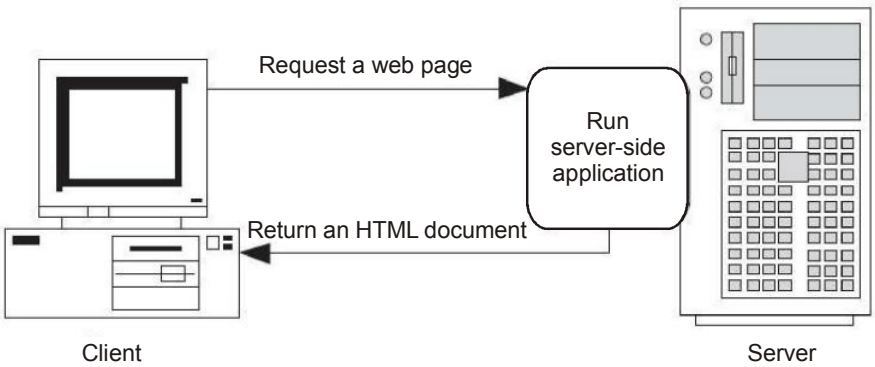
Figure 1.3: Server-side Architecture



Notes

To counter these problems, Microsoft created higher-level development platforms, such as ASP and ASP.NET. Both of these technologies allow developers to program dynamic web pages without worrying about the low-level implementation details. For that reason, both platforms have been incredibly successful.

Figure 1.4: Server-side Web Application



The original ASP platform garnered a huge audience of nearly one million developers, becoming far more popular than even Microsoft anticipated. It wasn't long before it was being wedged into all sorts of unusual places, including mission-critical business applications and highly trafficked e-commerce sites. Because ASP wasn't designed with these uses in mind, performance, security, and configuration problems soon appeared.

That's where ASP.NET comes into the picture. ASP.NET was developed as an industrial strength web application framework that could address the limitations of ASP. Compared to classic ASP, ASP.NET offers better performance, better design tools, and a rich set of readymade features. ASP.NET was wildly popular from the moment it was released—in fact, it was put to work in dozens of large-scale commercial websites while still in beta form.

Notes Despite having similar underpinnings, ASP and ASP.NET are radically different. ASP is a script-based programming language that requires a thorough understanding of HTML and a good deal of painful coding. ASP.NET, on the other hand, is an object-oriented programming model that lets you put together a web page as easily as you would build a Windows application.

1.2.1 Advantages of Server-side Programs

The various advantages of server-side programs are:

1. All programs reside in one machine called the Server. Any number of remote machines (called clients) can access the server programs.
2. New functionalities to existing programs can be added at the server-side which the clients' can advantage without having to change anything from their side.
3. Migrating to newer versions, architectures, design patterns, adding patches, switching to new databases can be done at the server-side without having to bother about clients' hardware or software capabilities.
4. Issues relating to enterprise applications like resource management, concurrency, session management, security and performance are managed by service-side applications.

Notes

5. They are portable and possess the capability to generate dynamic and user-based content (e.g. displaying transaction information of credit card or debit card depending on user's choice).

1.2.2 Types of Server-side Programs

Various types of server-side programs are:

1. Active Server Pages (ASP)
2. Java Servlets
3. Java Server Pages (JSPs)
4. Enterprise Java Beans (EJBs)
5. PHP

1.3 Client-side Programming

At the same time that server-side web development was moving through an alphabet soup of technologies, a new type of programming was gaining popularity. Developers began to experiment with the different ways they could enhance web pages by embedding miniature applets built with JavaScript, ActiveX, Java, and Flash into web pages. These client-side technologies don't involve any server processing. Instead, the complete application is downloaded to the client browser, which executes it locally.

The greatest problem with client-side technologies is that they aren't supported equally by all browsers and operating systems. One of the reasons that web development is so popular in the first place is because web applications don't require setup CDs, downloads, and other tedious (and error-prone) deployment steps. Instead, a web application can be used on any computer that has Internet access. But when developers use client-side technologies, they encounter a few familiar headaches. Suddenly, cross-browser compatibility becomes a problem. Developers are forced to test their websites with different operating systems and browsers, and they might even need to distribute browser updates to their clients. In other words, the client-side model sacrifices some of the most important benefits of web development.

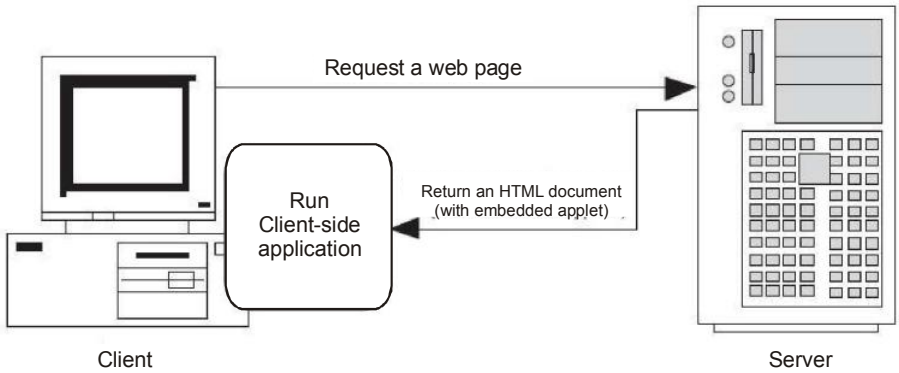
For that reason, ASP.NET is designed as a server-side technology. All ASP.NET code executes on the server. When the code is finished executing, the user receives an ordinary HTML page, which can be viewed in any browser. Figure 1.5 shows the difference between the server-side and the client-side model.

These are some other reasons for avoiding client-side programming:

1. ***Isolation:*** Client-side code can't access server-side resources. For example, a client-side application has no easy way to read a file or interact with a database on the server (at least not without running into problems with security and browser compatibility).
2. ***Security:*** End users can view client-side code. And once malicious users understand how an application works, they can often tamper with it.
3. ***Thin clients:*** As the Internet continues to evolve, web-enabled devices such as mobile phones, palmtop computers, and PDAs (personal digital assistants) are appearing. These devices can communicate with web servers, but they don't support all the features of a traditional browser. Thin clients can use server-based web applications, but they won't support client-side features such as JavaScript.

Notes

Figure 1.5: Client-side Web Application



However, client-side programming isn't truly dead. In many cases, ASP.NET allows you to combine the best of client-side programming with server-side programming. For example, the best ASP.NET controls can intelligently detect the features of the client browser. If the browser supports JavaScript, these controls will return a web page that incorporates JavaScript for a richer, more responsive user interface.

Table 1.1: Pros and Cons of Client-side Programming Tools

Type	Examples	Pros	Cons
Scripting Language	JavaScript PerlScript	Safe, in theory cannot access client files/hardware. Common to client and server.	Limited, cannot access client files/hardware. Little interactivity, cannot generate graphics, sound, etc.
Scripting Language	VBScript	Can access client files/hardware, extensible using ActiveX. Common to client and server.	Demands trust, OK perhaps for an intranet. Requires installation of ActiveX objects on client system.
Plug-in	Shockwave	Can extend browser capabilities.	Demands trust that the plug-in is safe.
Applet	Java	Safe, cannot access client files/hardware directly. General programming language with graphics, user interface, etc. Program delivered to client on demand by server.	Cannot access client files/hardware. High user interface development time, comparable to C++.

Task

HTML is a good and very simple language of internet programming but these days ASP more prefer than HTML. Specify the reason?

1.4 Common Language Runtime

The Common Language Runtime (CLR) is an Execution Environment. It works as a layer between Operating Systems and the applications written in .Net languages that conforms to the Common Language Specification (CLS). The main function of Common Language Runtime is to convert the Managed Code into native code and then execute the Program. The Managed Code compiled only when it needed, that is it converts the appropriate instructions when each function is called. The Common Language Runtime (CLR)'s Just In Time (JIT) compilation converts Intermediate Language (MSIL) to native code on demand at application run time.

Notes

During the execution of the program, the Common Language Runtime manages memory, Thread execution, Garbage Collection (GC), Exception Handling, Common Type System (CTS), code safety verifications, and other system services. The CLR (Common Language Runtime) defines the Common Type System, which is a standard type system used by all .Net languages. That means all .NET programming languages uses the same representation for common Data Types, so Common Language Runtime is a language-independent runtime environment. The Common Language Runtime environment is also referred to as a managed environment, because during the execution of a program it also controls the interaction with the Operating System. In the coming section you can see what are the main functions of Common Language Runtime.

The CLR is the engine that supports all the .NET languages. Many modern languages use runtimes. In VB 6, the runtime logic is contained in a DLL file named msvbvm60.dll. In C++, many applications link to a file named mscrt40.dll to gain common functionality. These runtimes may provide libraries used by the language, or they may have the additional responsibility of executing the code (as with Java).

Runtimes are nothing new, but the CLR is Microsoft's most ambitious runtime to date. Not only does the CLR execute code, it also provides a whole set of related services such as code verification, optimization, and object management.

Notes The CLR is the reason that some developers have accused .NET of being a Java clone. The claim is fairly silly. It's true that .NET is quite similar to Java in key respects (both use a special managed environment and provide features through a rich class library), but it's also true that every programming language "steals" from and improves on previous programming languages. This includes Java, which adopted parts of the C/C++ language and syntax when it was created. Of course, in many other aspects .NET differs just as radically from Java as it does from VBScript.

All .NET code runs inside the CLR. This is true whether you're running a Windows application or a web service. For example, when a client requests an ASP.NET web page, the ASP.NET service runs inside the CLR environment, executes your code, and creates a final HTML page to send to the client.

The implications of the CLR are wide-ranging:

1. ***Deep language integration:*** VB and C#, like all .NET languages, compile to IL. In other words, the CLR makes no distinction between different languages—in fact, it has no way of knowing what language was used to create an executable. This is far more than mere language compatibility; it's language integration.
2. ***Side-by-side execution:*** The CLR also has the ability to load more than one version of a component at a time. In other words, you can update a component many times, and the correct version will be loaded and used for each application. As a side effect, multiple versions of the .NET Framework can be installed, meaning that you're able to upgrade to new versions of ASP.NET without replacing the current version or needing to rewrite your applications.
3. ***Fewer errors:*** Whole categories of errors are impossible with the CLR. For example, the CLR prevents many memory mistakes that are possible with lower-level languages such as C++.

Along with these truly revolutionary benefits, the CLR has some potential drawbacks. Here are three issues that are often raised by new developers but aren't always answered:

1. ***Performance:*** A typical ASP.NET application is much faster than a comparable ASP application, because ASP.NET code is compiled to machine code before it's executed.

Notes

However, processor-crunching algorithms still can't match the blinding speed of well-written C++ code, because the CLR imposes some additional overhead. Generally, this is a factor only in a few performance-critical high-workload applications (such as real-time games). With high-volume web applications, the potential bottlenecks are rarely processor-related but are usually tied to the speed of an external resource such as a database or the web server's file system. With ASP.NET caching and some well-written database code, you can ensure excellent performance for any web application.

2. ***Code transparency:*** IL is much easier to disassemble, meaning that if you distribute a compiled application or component, other programmers may have an easier time determining how your code works. This isn't much of an issue for ASP.NET applications, which aren't distributed but are hosted on a secure web server.

3. ***Questionable cross-platform support:*** No one is entirely sure whether .NET will ever be adopted for use on other operating systems and platforms. Ambitious projects such as Mono (a free implementation of .NET on Linux, Unix and Windows) are currently underway (www.mono-project.com). However, .NET will probably never have the wide reach of a language such as Java because it incorporates too many different platform-specific and operating system–specific technologies and features.

1.5 .Net Class Library

The Formatter and FileStream classes are just two of more than 2,500 classes in the .NET Framework that provide plumbing and system services for .NET applications. Some of the functionality provided by the .NET Framework includes

1. Base class library (basic functionality such as strings, arrays, and formatting)
2. Networking
3. Security
4. Remoting
5. Diagnostics
6. I/O
7. Database
8. XML
9. Web services that allow us to expose component interfaces over the Internet
10. Web programming
11. Windows user interface

1.6 Role of .Net Class Libraries

The .NET Framework class library is a library of classes, interfaces, and value types that are included in the Windows Software Development Kit (SDK). This library provides access to system functionality and is designed to be the foundation on which .NET Framework applications, components, and controls are built.

The .Net Framework class library (FCL) provides the core functionality of .Net Framework architecture. The .Net Framework Class Library includes a huge collection of reusable classes, interfaces, and value types that expedite and optimize the development process and provide access to system functionality.

Notes

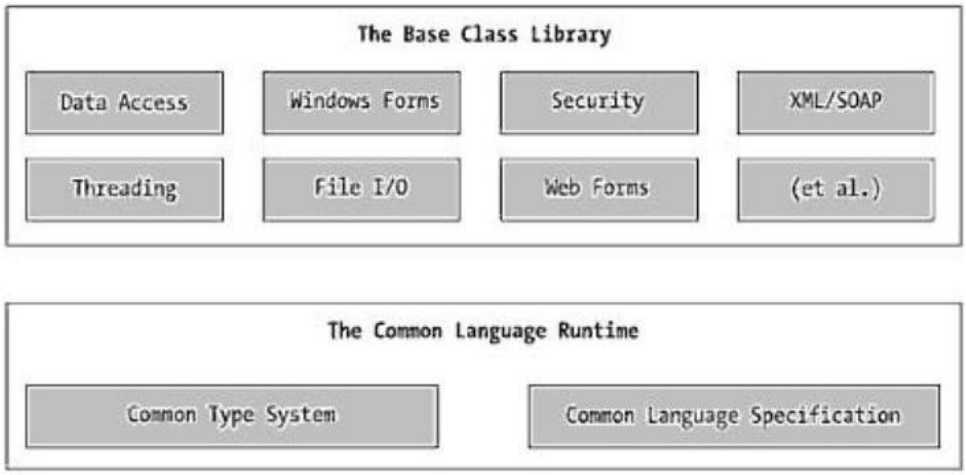
The .Net Framework class library organized in a hierarchical tree structure and it is divided into Namespaces. Namespaces is a logical grouping of types for the purpose of identification. Framework class library provides the consistent base types that are used across all .NET enabled languages. The Classes are accessed by namespaces, which reside within Assemblies. The System Namespace is the root for types in the .NET Framework. The .Net Framework class library classes are managed classes that provide access to System Services. The .Net Framework class library classes are object oriented and easy to use in program developments. Moreover, third-party components can integrate with the classes in the .NET Framework.

In addition to the CLR and CTS/CLS specifications, the .NET platform provides a base class library that is available to all .NET programming languages. Not only does this case class library encapsulate various primitives such as threads, file input/output (I/O), graphical rendering, and interaction with various external hardware devices, but it also provides support for a number of services required by most real-world applications.

Example: The base class libraries define types that facilitate database access, XML manipulation, programmatic security, and the construction of web-enabled front ends.

From a high level, you can visualize the relationship between the CLR, CTS, CLS and the base class library as shown in Figure 1.6.

Figure 1.6: The CLR, CTS, CLS and Base Class Library Relationship



Case Study

Semantic Web with Global TeleGenetics and Telerik

When Carl Taswell looked closely at the idea of the semantic web, he saw there were not yet adequate frameworks or other underlying foundational elements available to build needed semantic web infrastructures. Seizing the opportunity to make a lasting impact, he and his team embarked on an ambitious undertaking – creating a whole system for metadata management that could then be used as part of the architecture of the semantic web.

In the same year that Sir Tim Berners-Lee, creator of the World Wide Web, was offering his vision of its future as a semantic web, Carl Taswell was busy founding the company that would one day play a role in contributing to that vision. Taswell, who good-naturedly

Contd...

Notes

refers to himself as the "Chief Cook and Bottle-Washer" and "Head Square" at the life sciences informatics company, quickly grasped the significance and potential benefit of the proposed "semantic web". He saw it could have extraordinary value in medical research and clinical healthcare settings, where a freer flow of information and better knowledge sharing could result in swifter discoveries and more effective treatment plans.

"It's about ensuring communication and interoperability, and the improvements it can bring to healthcare. The goal of implementing a cyberinfrastructure capable of effective data integration, communication, and interoperability across different specialty domains in healthcare and life sciences is a major challenge that has yet to be solved," he says. "But we believe it can and must be addressed – neurology should be able to talk to genetics, genetics should be able to talk to nuclear medicine, nuclear medicine should be able to talk to cardiology. Opening the door to greater cross-disciplinary understanding, interaction, collaboration, and cooperation means a better standard of care for all of us."

Yet, when Taswell looked closely at the idea of the semantic web, he saw there were not yet adequate frameworks or other underlying foundational elements available to build needed semantic web infrastructures. Seizing the opportunity to make a lasting impact, he and his team embarked on an ambitious undertaking – creating a whole system for metadata management that could then be used as part of the architecture of the semantic web.

The project would combine two key constructs: The "Problem Oriented Registry of Tags And Labels" (PORTAL) and the "Domain Ontology Oriented Resource System" (DOORS), or the "PORTAL-DOORS System" (PDS) for short. But first, Taswell's team would need to find the tools and resources necessary to bring life to their semantic web project.

Weaving a Better Web

The goal of the PORTAL-DOORS Project (PDP) was to provide both anonymous user read access and authenticated agent write access to resource metadata in the PDS registries and directories, as well as offering detailed information about the ongoing design, development, and implementation of PDS as a cyberinfrastructure for the semantic web. But crafting this resilient foundation called for sturdy, flexible tools capable of meeting PDP's exacting needs. Upon researching available commercial toolsets, Taswell and his team settled on the Telerik Ultimate Collection, a rich collection of .NET development tools.

"If we really wanted to figure out how to make this work out over the long-term, we first had to figure out what are we were going to do to make PDP successful, and how to overcome the barriers and hurdles standing in our way," recalls Taswell. "We asked ourselves what we could use to make sure we're going to succeed, and the answer turned out to be Telerik's tools."

The Telerik Ultimate Collection incorporates a comprehensive collection of developer productivity technologies, such as RadControls. In particular, Taswell and his team relied heavily on use of grids from RadControls for ASP.NET AJAX, Extensions for ASP.NET MVC, and RadControls for Silverlight throughout the development cycle. With the combined toolsets, developers were easily able to create both the web interfaces for anonymous users of metadata records at the read-only registries and directories, and the web interfaces for authorized owner-authors of the metadata records at the read-write registrars.

With PDP being created using multiple frameworks resulting in a variety of versions, the Telerik Ultimate Collection was an ideal fit for the project. The all-encompassing compilation provided access to the full range of Telerik control suites, delivering effortless

Contd...

Notes

integration of components such as RadMenu for ASP.NET AJAX, and enabling the binding of Telerik's Extensions for ASP.NET MVC to AJAX to be completed swiftly and easily.

The great benefit of using the Telerik UI Suite was the ability to maintain a professional look-and-feel while continuing an ongoing R&D project. Thus, with Telerik's tools, the PDP team was able to create the metadata management frameworks needed to bring the PORTAL-DOORS System to life, while saving time and preserving essential resources.

"When you're dealing with limited funds and personnel, you have to seek out the easiest and most efficient way of making the most of what you have, while still making it look reasonably professional," notes Taswell. "We're not trading stocks on Wall Street and we're not a Fortune 500 corporation, so the problem becomes, how do you get the most out of the time, resources, and effort you put in? I'm convinced it's a solution like Telerik's controls. Then, when you add in their commitment to customer service, you have a pretty good package for a small business that's working on making its mark in the world, like Global TeleGenetics."

A PORTAL-DOOR to Success:

PDS has become what Taswell's team envisioned: a robust, semantic-web ready application enabling users to perform customized data searches based on industry or specializations, which then generates more intelligent, meaningful, and relevant results.

By leveraging the time- and effort-saving technologies of the Telerik Ultimate Collection, the PDP has been able to move PDS from a conceptual architectural design to actual implemented code with working prototypes of registries and directories in a variety of different health care and life science specialty areas. Much of the foundational architecture has been completed and the project is moving closer to being opened up to use for other specialty areas in the medical research and healthcare community. Taswell and the PDP team will be transferring record editing and management responsibilities to other organizations expected to make the greatest use of the system. When fully populated, PORTAL-DOORS will facilitate better knowledge sharing and collaboration across different specialty areas.

1.7 Summary

- .NET is a moving target and is clearly in a state of flux. While much of what we know about .NET is not likely to change, new methodologies and paradigms are sure to arise that will add to what we already know and understand.
- One of .NET's goals, to make all web applications available on any device, requires decoupling data and business rules from the client and supporting them in the business and data service tiers.
- Microsoft's implementation of Web Services with SOAP is an industry-standard implementation with a focus on cross-platform integration.
- Web Services provide a mechanism for server-to-server communication on behalf of the user rather than redirecting the user to another site. Deployment is made easier with CLR, the Common Language Runtime. .NET's goals are manageability, scalability, flexibility, and reliability.
- All the Microsoft .NET languages use a common language runtime, which solves the problem of installing separate runtime for each of the programming languages. Microsoft .NET Common Language Runtime installed on a computer can run any language that is Microsoft .NET compatible.

Notes

1.8 Keywords

.NET Framework Class Library (FCL): The foundation of classes, interfaces, value types, services and providers that are used to construct .NET Framework desktop and Web-based (i.e., ASP.NET) applications.

.NET Framework: A programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services.

HTML (HyperText Markup Language): A document-layout and hyperlink-specification language. HTML is used to describe how the contents of a document (e.g., text, images, and graphics) should be displayed on a video monitor or a printed page. HTML also enables a document to become interactive with other documents and resources by using hypertext links embedded into its content. HTML is the standard content display language of the World Wide Web (WWW), and is typically conveyed between network hosts using the HTTP protocol.

HTTP (Hyper Text Transfer Protocol): An Internet protocol used to transport content and control information across the World Wide Web (WWW). Web content typically originates from Web servers (also called HTTP servers) that run services which support the HTTP protocol. Web clients (i.e., Web browsers) access the content on the server using the rules of the HTTP protocol. The actual Web content is encoded using the HTML or XHTML languages.

Interface Definition Language (IDL): A language used to describe object interfaces by their names, methods, parameters, events, and return types. A compiler uses the IDL information to generate code to pass data between machines. Microsoft's IDL, called COM IDL, is compiled using the Microsoft IDL compiler (MIDL). MIDL generates both type libraries and proxy and stub code for marshaling parameters between COM interfaces.

Namespace: A logical grouping of the names (i.e., identifiers) used within a program. A programmer defines multiple namespaces as a way to logically group identifiers based on their use.

1.9 Self Assessment

Choose the appropriate answers:

1. HTML stands for:
   - (a) High Text Markup Language
   - (b) Hyper Text Markup Language
   - (c) High Text Maker Language
   - (d) Hyper Text Maker Language
2. URL stands for:
   - (a) Uniform Resource Locator
   - (b) Uniform Resource Location
   - (c) Union Resource Locator
   - (d) Union Regional Locator

Notes

3. Types of HTML pages:
   (a) Single
   (b) 2
   (c) 4
   (d) 5
4. CGI stands for:
   (a) Common Group Interface
   (b) Common Goal Interface
   (c) Common Gateway Interface
   (d) Common Gateway Interchange

State whether the following statements are True or False:

5. ASP.NET is designed as a server-side technology.
6. The Common Language Runtime (CLR) is not an Execution Environment.
7. The .Net Framework class library (FCL) provides the core functionality of .Net Framework architecture.
8. HTML forms do not allow web application developers to design standard input pages.
9. A basic HTML page is a little like a word-processing document.
10. HTML consists of tags and data.
11. An HTML file is a text file containing small markup tags.

1.10 Review Questions

1. Explain .Net framework.
2. Explain the features of .Net framework.
3. What is base class library?
4. Describe the roles of common intermediate language.
5. What do you mean by additional .NET programming languages?
6. Describe HTML form.
7. Differentiate between server-side and client-side programming.
8. What do you mean by .Net class library?
9. Describe how will you create a web page in HTML programming with two radio button and two check boxes.
10. Describe pros and cons of client-side programming.

Notes

Answers: Self Assessment

1.	(b)	2.	(a)
3.	(b)	4.	(c)
5.	True	6.	False
7.	True	8.	False
9.	True	10.	True
11.	True		

1.11 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.Net

www.webopedia.com

Notes

Unit 2: Visual Studio

CONTENTS

Objectives

Introduction

2.1 Creating Websites

2.1.1 Step by Step Website Creation

2.1.2 Websites and Web Projects

2.1.3 The Hidden Solution Files

2.1.4 The Solution Explorer

2.1.5 Adding Web Forms

2.1.6 Migrating a Website from a Previous Version of Visual Studio

2.2 Designing a Webpage

2.2.1 Adding Web Controls

2.2.2 The Properties Window

2.3 The Anatomy of a Web Form

2.3.1 The Web Form Markup

2.3.2 The Page Directive

2.3.3 The Doctype

2.3.4 The Essentials of XHTML

2.3.5 A Complete Web Page

2.4 Writing Code

2.4.1 The Code-Behind Class

2.4.2 Adding Event Handlers

2.4.3 IntelliSense and Outlining

2.5 Visual Studio Debugging

2.5.1 The Visual Studio Web Server

2.5.2 Single-step Debugging

2.5.3 Variable Watches

2.6 Summary

2.7 Keywords

2.8 Self Assessment

2.9 Review Questions

2.10 Further Readings

Notes

Objectives

After studying this unit, you will be able to:

- Explain Website creation
- Describe designing a webpage
- Know anatomy of a web form
- Explain visual studio debugging

Introduction

In the ancient days of web programming, developers created web pages with simple text editors such as Notepad. Other choices were available, but each suffered from its own quirks and limitations. The standard was a gloves-off approach of raw HTML with blocks of code inserted wherever necessary.

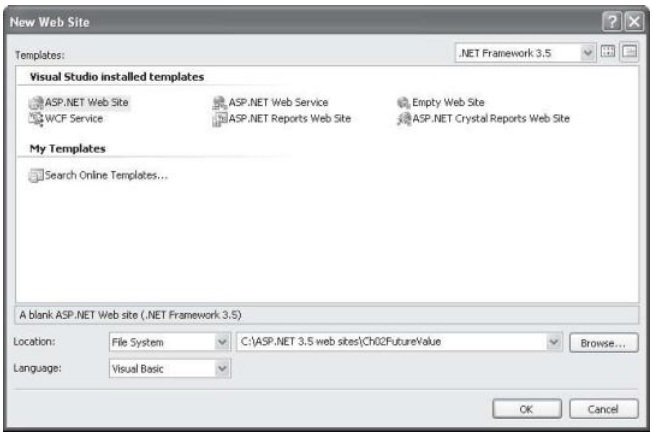
Visual Studio changes all that. First, it's extensible and can even work in tandem with other straight HTML editors such as Microsoft Expression Web or Adobe Dreamweaver. In other words, you can do the heavy-duty coding with Visual Studio, but use another web design tool to make everything look pretty. Second, Visual Studio includes indispensable time-saving features. For example, it gives you the ability to drag and drop web pages into existence and troubleshoot misbehaving code. Visual Studio even includes a built-in test web server, which allows you to create and test a complete ASP.NET website without worrying about web server settings.

In this unit, you'll learn how to create a web application using Visual Studio. Along the way, you'll take a look at the anatomy of an ASP.NET web form, and review the essentials of XHTML. You'll also learn how IntelliSense can dramatically reduce the number of errors you'll make, and how to use Visual Studio's legendary single-step debugger to look under the hood and "watch" your program in action. By the end of this unit, you'll be well acquainted with the most important tool in any ASP.NET developer's toolkit (Visual Studio) and you'll understand the basic principles of web development with ASP.NET.

2.1 Creating Websites

In Visual Studio 2008, a web application is called a web site, and Figure 2.1 shows the dialog box for starting a new web site. After you open the New Web Site dialog box, you select the language you want to use for the web site and you specify the location where the web site will be created.

Figure 2.1



Notes

The Location drop down list gives you three options for specifying the location of the web site. The simplest method is to create a file-system web site. This type of web site can exist in any folder on your local hard disk, or in a folder on a shared network drive. You can run a file-system web site using either Visual Studio's built-in development server or IIS.

You use the HTTP option to create a web site that runs under IIS on your local computer or on a computer that can be accessed over a local area network. To use this option, you must specify the IIS server where you want to create the web site. In addition, you must select or create the IIS directory that will contain the files for the web site, or you must select or create a virtual directory for the web site.

The third option, FTP, lets you create a web site on a remote server by uploading it to that server using FTP. To create this type of web site, you must specify at least the name of the FTP server and the folder where the web site resides.

In addition to the language and location, Visual Studio 2008 lets you choose a target framework for a web site. By default, .NET Framework 3.5 is used, which is usually what you want. Then, you can use the features that this framework provides within your web applications. If you'll be deploying the application to a server that doesn't have .NET Framework 3.5, however, you may want to target .NET Framework 2.0 or 3.0 instead. Then, you can be sure that you'll use only the features that these frameworks provide.

By default, Visual Studio 2008 creates a solution file for your web site in My Documents\Visual Studio 2008\Projects. This solution file is stored in this folder regardless of the location of the web site itself. To change the location where solutions are stored by default, choose Tools'! Options. Then, expand the Projects and Solutions node, select the General category, and enter the location in the Visual Studio Projects Location text box.

In the dialog box in this example, I'm starting a new file-system web site named Ch02FutureValue in the ASP.NET 3.5 web sites folder on my own PC. Then, when you click the OK button, Visual Studio creates the folder name.

Table 2.1: Location Options for ASP.NET Websites

Option	Description
File System	A web site created in a folder on your local computer or in a shared folder on a network. You can run the web site directly from the built-in development server or create an IIS virtual directory for the folder and run the application under IIS.
HTTP	A web site created under the control of an IIS web server. The IIS server can be on your local computer or on a computer that's available over a local area network.
FTP	A web site created on a remote hosting server.

2.1.1 Step by Step Website Creation

1. To create a new ASP.NET web application, called a web site, you use the File → New Web Site command.
2. A web project is a project that's used for the development of a web site. In practice, web sites are often referred to as web projects, and vice versa. However, ASP.NET 3.5 web sites don't use project files. Instead, they use web.config files to store project information.
3. When you start a new web site, Visual Studio creates a solution file for the web site in the default location for solution files, which is normally My Documents\Visual Studio 2008\Projects.

Notes

4. By default, the web sites you create target .NET Framework 3.5 so you can use the features it provides. If you want to target a different version of the .NET Framework, you can select the version from the drop-down list in the upper right corner of the dialog box.

5. Visual Studio 2008 provides for a new type of web project, called a web application project. With this type of project, you can create web applications that are similar in structure to applications created using Visual Studio 2003.

Ch02FutureValue and puts the starting files for the web site in that folder. It also creates a solution file in the default folder for those files.

The folders and files that are used for developing a web site can be referred to as a web project. So in practice, web sites are often referred to as web projects, and vice versa. It's important to note, however, that ASP.NET 3.5 web sites don't use project files to store project information. Instead, they use web.config files.

2.1.2 Websites and Web Projects

Ordinarily, Visual Studio uses project files to store information about the applications you create. Web applications are a little unusual because Visual Studio doesn't necessarily create project files for them. In fact, if you followed the steps in the previous section, you created a new website with no project file.

This system, which is called projectless development, is different from the way Visual Studio works with other types of applications, such as stand-alone components and Windows programs. It's designed to keep your website directory clean and uncluttered, and thereby simplify the deployment of your web application. This way, when it's finally time to upload your website to a live web server, you can copy the entire folder without worrying about excluding files that are only used for development purposes. Projectless development is also handy if you're programming with a team of colleagues, because you can each work on separate pages without needing to synchronize project and solution files.

For most web developers, this is all you need to know. However, there's actually another option: project-based development, or web projects. Web projects are the older way of creating ASP.NET web applications, and they're still supported in Visual Studio 2008 for use in specific scenarios.

You can create a web project by choosing File → New → Project, and then choosing the ASP.NET Web Application template. Web projects support all the same features as projectless websites, but they use an extra project file (with the extension .csproj). The web project file keeps track of the web pages, configuration files, and other resources that are considered part of your web application. It's stored in the same directory as all your web pages and code files. Essentially, there are just a few reasons why you would consider using web projects:

1. You have an old web project that was created in a version of Visual Studio before Visual Studio 2005. When you open this project in Visual Studio 2008, it will be migrated as a web project automatically to avoid strange compatibility quirks that might otherwise crop up.

2. You want to place two (or more) web projects in the same website folder. Technically, ASP.NET will consider these two projects to be one web application. However, with web projects, you have the flexibility to work on the files separately in Visual Studio. You simply add the files that you want to group together to your project.

3. You have a really huge website that has lots of resources files (for example, thousands of images). Even though these files are a part of your website, you might not want them to appear in the Solution Explorer window in Visual Studio because they can slow down the

Notes

development environment. If you use web projects, you can easily get around this issue just don't add these resource files to your project.

4. You are using the MSBuild utility to create an automated deployment process. The MSBuild utility uses project files. For example, a large company might devise a build strategy that automatically signs compiled web application files and deploys them to a production web server. MSBuild isn't discussed in this book, but you can find more information by looking up the "MSBuild" entry (with that capitalization) in the index of the Visual Studio Help.

2.1.3 The Hidden Solution Files

Visual Studio allows you to create ASP.NET applications without project files. However, you might be surprised to learn that Visual Studio still creates one type of resource file, called a solution file. Solutions are a similar concept to projects the difference is that a single solution can hold one or more projects. Whenever you're working in Visual Studio, you're working with a solution. Often, that solution contains a single projectless website, but in more advanced scenarios it might actually hold additional projects, such as a web application and a component that you're using with your website.

At this point, you're probably wondering where Visual Studio places solution files. It depends, but in a typical projectless web application, Visual Studio quietly tucks the solution files away into the user-specific document directory. In Windows Vista, you'll find it in a directory that's named in this form:

c:\Users\[UserName]\Documents\Visual Studio 2008\Projects\[WebsiteFolderName]

In earlier versions of Windows, the directory is named in this form:

c:\Documents and Settings\[UserName]\My Documents\Visual Studio 2008\Projects\[WebsiteFolderName]

Either way, this system can get a bit confusing, because the rest of your website files will be placed in a completely different directory.

Each solution has two solution files, with the file extensions .sln and .suo.

SampleSite.sln

SampleSite.suo

When you open a previously created website, Visual Studio locates the matching solution file automatically, and uses the settings in that solution.

The solution files store some Visual Studio specific details that aren't directly related to ASP.NET, such as debugging and view settings.

Example: Visual Studio tracks the files that are currently open so it can reopen them when you resume your website development.

The solution files aren't essential. In fact, if you move your website to another computer (or just place them in another location on your computer), Visual Studio won't be able to locate the original solution files, and you'll lose the information they store. You'll can also run into trouble if you create two websites with the same name in different locations, in which case the newer solution files may overwrite the older ones. However, because the information in the solution files isn't really all that important, losing it isn't a serious problem. The overall benefits of a projectless system are usually worth the trade-off.

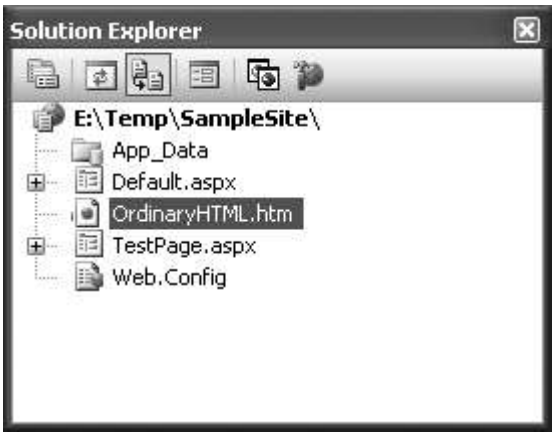
Notes

Usually, you can forget about solutions altogether, and let Visual Studio manage them seamlessly. But in some cases, you might want to keep a closer eye on your solution files so you can use them later. For example, you might want to use a solution file to open up a combination of projects that you're working on at the same time.

2.1.4 The Solution Explorer

To take a high-level look at your website, you can use the Solution Explorer the window at the top-right corner of the design environment that lists all the files in your web application directory (Figure 2.2).

Figure 2.2: The Solution Explorer



The Solution Explorer reflects everything that's in the application directory of a projectless website. No files are hidden. This means if you add a plain HTML file, a graphic, or a subdirectory in Windows Explorer, the next time you fire up Visual Studio you'll see the new contents in the Solution Explorer. If you add these same ingredients while Visual Studio is open, you won't see them right away. Instead, you'll need to refresh the display. To do so, right-click the website folder in the Solution Explorer (which appears just under the Solution item at the top of the tree) and choose Refresh Folder.

Of course, the whole point of the Solution Explorer is to save you from resorting to using Windows Explorer. Instead, it allows you to perform a variety of file management tasks within Visual Studio. You can rename, delete, or copy files by simply right-clicking the item and choosing the appropriate command.

2.1.5 Adding Web Forms

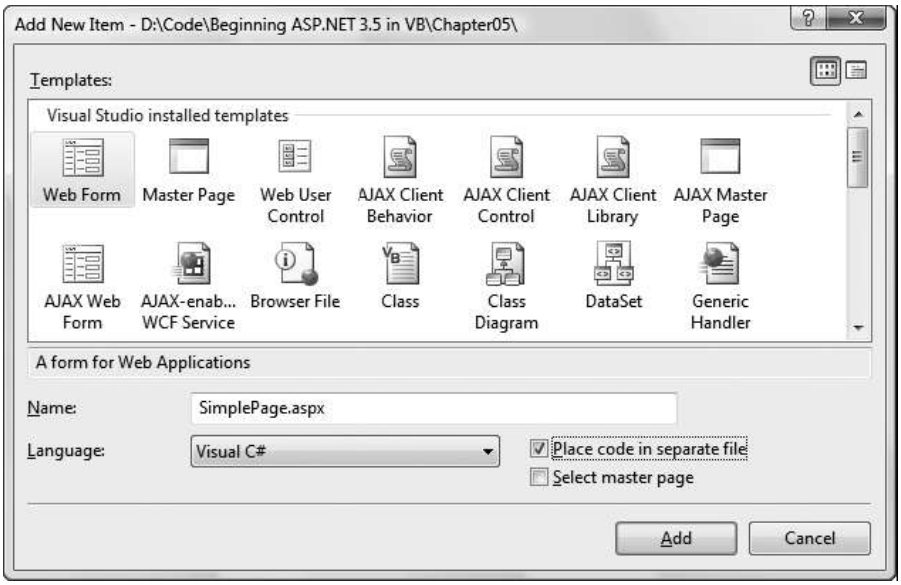
As you build your website, you'll need to add new web pages and other items. To add these ingredients, choose Website → Add New Item from the Visual Studio menu. When you do, the Add New Item dialog box will appear.

You can add various types of files to your web application, including resources you want to use (such as bitmaps), ordinary HTML files, code files with class definitions, style sheets, data files, configuration files, and much more. Visual Studio even provides basic designers that allow you to edit most of these types of files directly in the IDE. However, the most common ingredients that you'll add to any website are web forms ASP.NET web pages that are fueled with C# code. Your website begins with one (named Default.aspx), but you're sure to need many more before your application is complete.

Notes

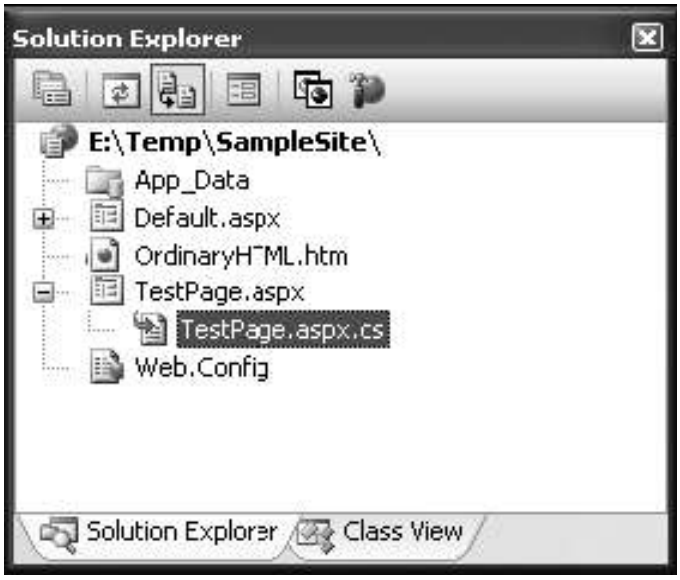
To add a web form, choose Web Form in the Add New Item dialog box. You'll see two new options at the bottom of the Add New Item dialog box (Figure 2.3).

Figure 2.3: Adding an ASP.NET Web Form



The Place Code in a Separate File option allows you to choose the coding model for your web page. If you clear this check box, Visual Studio will create a single-file web page. You must then place all the C# code for the file in the same file that holds the HTML markup. If you select the Place Code in a Separate File option, Visual Studio will create two distinct files for the web page, one with the markup and the other for your C# code. This is the more structured approach that you'll use in this book. The key advantage of splitting the web page into separate files is that it's more manageable when you need to work with complex pages. However, both approaches give you the same performance and functionality.

Figure 2.4: A Code File for a Web Page



Notes

You'll also see another option named Select Master Page, which allows you to create a page that uses the layout you've standardized in a separate file.

Once you've chosen the coding model and typed in a suitable name for your web page, click Add to create it. If you've chosen to use the Place Code in Separate File check box (which is recommended), your project will end up with two files for each web page. One file includes the web page markup (and has the file extension .aspx). The other file stores the source code for the page (and uses the same file name, with the file extension .aspx.cs). To make the relationship clear, the Solution Explorer displays the code file underneath the .aspx file (Figure 2.4).

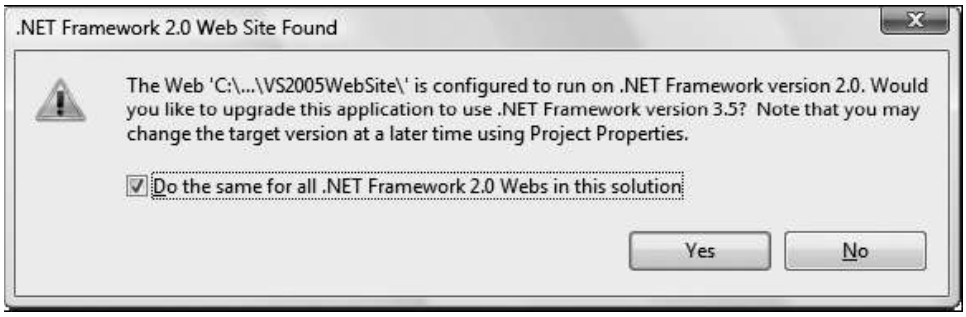
You can also add files that already exist by selecting Website → Add Existing Item. You can use this technique to copy files from one website to another. Visual Studio leaves the original file alone and simply creates a copy in your web application directory. However, don't use this approach with a web page that has been created in an older version of Visual Studio. Instead, refer to the following section to convert your old application and bring it into Visual Studio 2008.

2.1.6 Migrating a Website from a Previous Version of Visual Studio

If you have an existing ASP.NET web application created with an earlier version of Visual Studio, you can migrate it to the ASP.NET world with ease.

If you created a projectless website with Visual Studio 2005, you use the File → Open → Web Site command, just as you would with a website created in Visual Studio 2008. The first time you open a Visual Studio 2005 website, you'll be asked if you want to adjust it to use ASP.NET 3.5 (Figure 2.5). If you choose Yes, Visual Studio makes a few simple changes to the web.config configuration file, so that the application can use .NET 3.5. If you choose No, your website will stay as it is, and it will continue targeting ASP.NET 2.0. (You can modify this detail at any time by choosing Website → Start Options.) Either way, you won't be asked again the next time you open the website, because your preference will be recorded in the hidden solution file that's stored in a user-specific Visual Studio directory.

Figure 2.5: Opening a Projectless Website that was created with Visual Studio 2005



If you created a web project with Visual Studio 2005, Visual Studio 2003, or Visual Studio .NET, you need to use the File → Open → Project/Solution command. When you do, Visual Studio begins the Conversion Wizard. The Conversion Wizard is exceedingly simple. It prompts you to choose whether to create a backup and, if so, where it should be placed (Figure 2.6). If this is your only copy of the application, a backup is a good idea in case some aspects of your application can't be converted successfully. Otherwise, you can skip this option.

When you click Finish, Visual Studio performs an in-place conversion, which means it overwrites your web page files with the new versions. Any errors and warnings are added to a conversion log, which you can display when the conversion is complete.

Notes

Figure 2.6: Importing a Web Project that was created with an Older Version of Visual Studio

Task

Suppose you design a web application related to student details of your section. Now I want to add one more form in this application. Specify the web form addition in the application.

2.2 Designing a Webpage

Now that you know how to create a web site, you're ready to learn how to build a webpage. I'll show you the webpage that you're going to build. Then, I'll show you how to build it.

Visual Studio gives you three ways to look at an .aspx page:

1. ***Design view:*** Here you'll see a graphical representation of what your page looks like.
2. ***Source view:*** Here you'll see the underlying markup, with the HTML for the page and the ASP.NET control tags.
3. ***Split view:*** This combined view allows you to see both the design view and source view at once, stacked one on top of the other. This is the view that most ASP.NET developers prefer, provided they have enough screen space.

You can switch between these three views freely by clicking the Design, Split, and Source buttons at the bottom of the designer window.

You'll spend some time in the source view a bit later in this unit, when you dig into the web page markup. But first, it's easiest to start with the friendlier design view, and start adding content to your page.

Notes

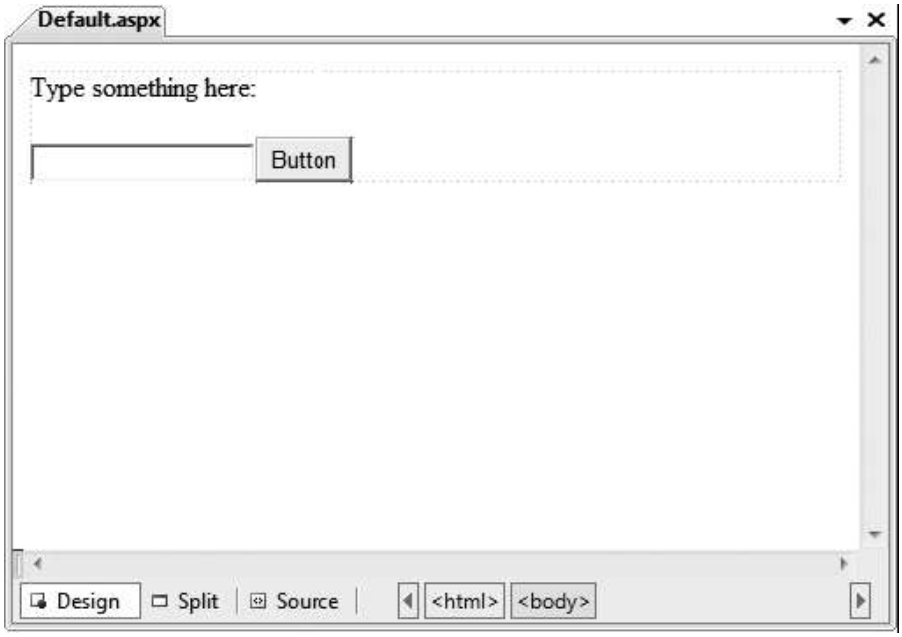
2.2.1 Adding Web Controls

To add an ASP.NET web control, drag the control you want from the Toolbox on the left and drop it onto your web page. Technically speaking, you can drop your controls onto a design view window or onto a source view window. However, it's usually easiest to position the control in the right place when you use design view. If you drop controls carelessly into the source view, they might not end up in the <form> section, which means they won't work in your page.

The controls in the Toolbox are grouped in numerous categories based on their functions, but you'll find basic ingredients such as buttons, labels, and text boxes in the Standard tab.

In a web form, controls are positioned line by line, like in a word processor document. To add a control, you need to drag and drop it to an appropriate place. To organize several controls in design view, you'll probably need to add spaces and hard returns (just hit Enter) to position elements the way you want them. Figure 2.7 shows an example with a TextBox, a Label, and a Button control.

Figure 2.7: The Design View for a Page

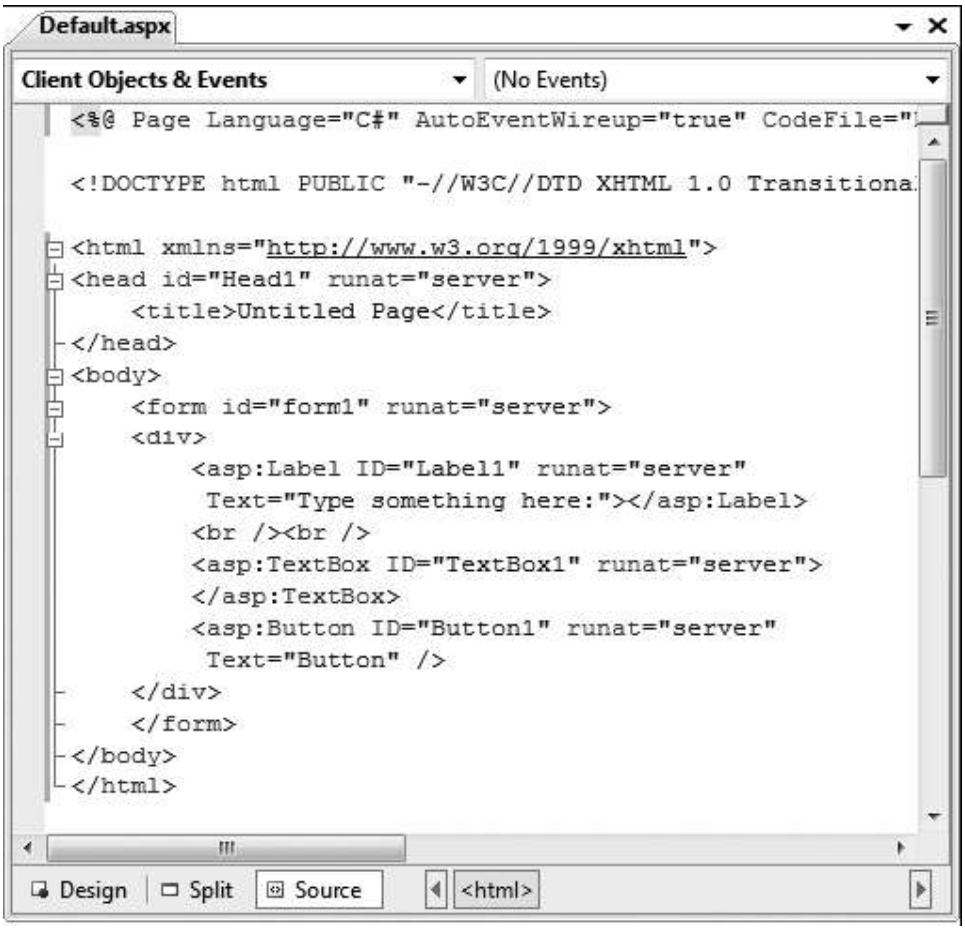


You'll find that some controls can't be resized. Instead, they grow or shrink to fit the amount of content in them. For example, the size of a Label control depends on how much text you enter in it. On the other hand, you can adjust the size of a Button or a TextBox control by clicking and dragging in the design environment.

As you add web controls to the design surface, Visual Studio automatically adds the corresponding control tags to your .aspx file. To look at the markup it's generated, you can click the Source button to switch to source view (or click the Split button to see both at once). Figure 2.8 shows what you might see in the source view for the page displayed in Figure 2.7.

Notes

Figure 2.8: The Source View for a Page



Using the source view, you can manually add attributes or rearrange controls. In fact, Visual Studio even provides IntelliSense features that automatically complete opening tags and alert you if you use an invalid tag. Whether you use the design or source view is entirely up to you Visual Studio keeps them both synchronized.

Notes When you use split view, Visual Studio keeps the source and design area synchronized. However, this synchronization process isn't always instantaneous. Generally, if you make a change in the design region, the source region is updated immediately. However, if you make a change in the source region, a yellow message bar appears at the top of the design region to warn you that the content has changed. To refresh the design region, simply click anywhere inside the design region. To refresh the design region without taking your fingers off the keyboard, press Ctrl+S to save the web page.

2.2.2 The Properties Window

Once you've added a web control to a web page, you'll probably want to tweak it a bit. For example, you might want to change the text in the button, the color of a label, and so on. Although you can make all your changes by editing the source markup by hand, Visual Studio

Notes

provides an easier option. Just under the Solution Explorer, in the bottom-right corner of the Visual Studio window, you'll see the Properties window, which shows you the properties of the currently selected web control and lets you tweak them.

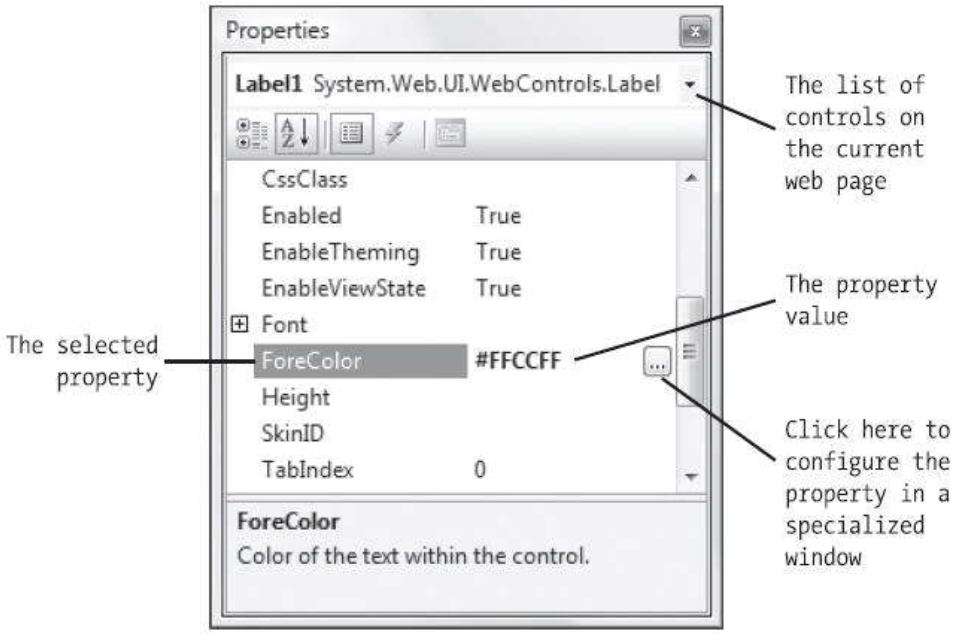
To configure a control in design view you must first select it on the page (either click it once in the design view or click somewhere inside the tag for that control in the source view). You'll know the right control is selected when you see its name appear in the drop-down list at the top of the Properties window. Alternatively, you can select your control by picking its name from the Properties window list.

Once you've selected the control you want you can modify any of its properties. Good ones to try include Text (the content of the control), ID (the name you use to interact with the control in your code), and ForeColor (the color used for the control's text).

Notes If the Properties window isn't visible, you can pop it into view by choosing View → Properties Window.

Every time you make a selection in the Properties window, Visual Studio adjusts the web page markup accordingly. Visual Studio even provides special "choosers" that allow you to select extended properties. For example, if you select a color property (such as ForeColor or BackColor) in the Properties window, a button with three dots (...) will appear next to the property, as shown in Figure 2.9.

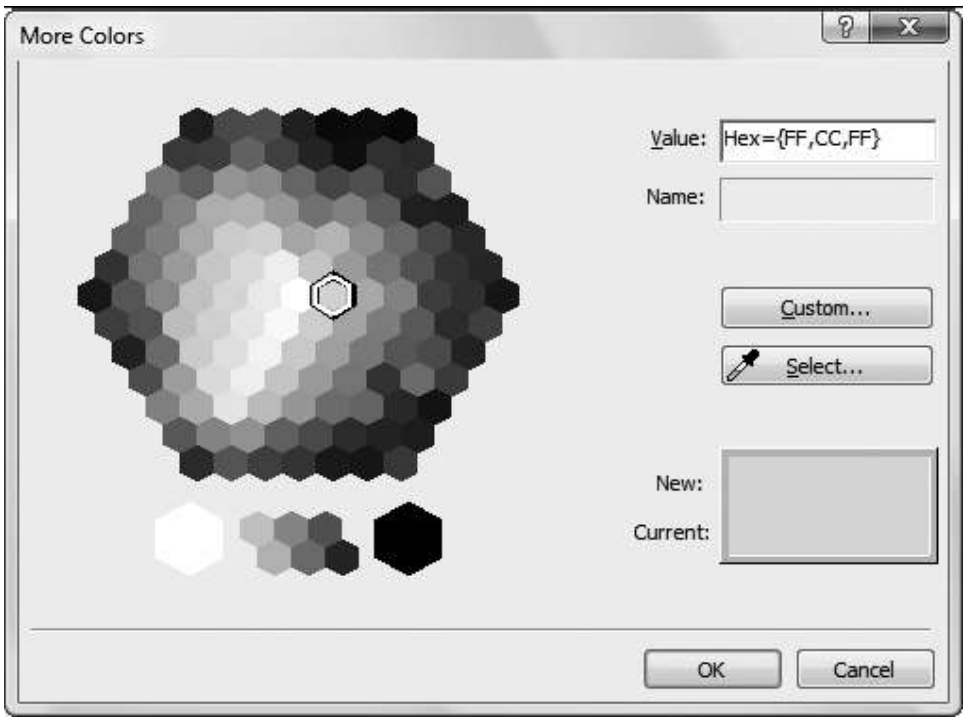
Figure 2.9: The ForeColor Property in the Properties Window



If you click this button, Visual Studio will show a dialog box where you can pick a custom color (Figure 2.10). Once you make your selection and click OK, Visual Studio will insert the HTML color code into the Properties window and update your web page markup.

Notes

Figure 2.10: Choosing a Color Value



You'll see a similar feature when configuring fonts. First, select the appropriate control on the design surface (or in the list at the top of the Properties window). Next, expand the Font property in the properties window and select the Name subproperty. A drop-down arrow will appear next to the property. Click this to choose from a list of all the fonts that are installed on your computer.

Along with web controls, you can also select ordinary HTML tags in the Properties window. However, there's a lot less you can do with them. Ordinary HTML tags aren't live programming objects, so they don't have nifty features that you can control. And ordinary HTML tags don't offer many options for formatting. If you want to change the appearance of a tag, your best choice is to create a style for your control.

Finally, you can select one object in the Properties window that needs some explanation the DOCUMENT object, which represents the web page itself. Using this object, you can set various options for the entire page, including the title that will be displayed in the browser, linked style sheets, and support for other features that are discussed later in this book (such as tracing and session state).

2.3 The Anatomy of a Web Form

So far, you've spent most of your time working with the web page in design view. There's nothing wrong with that after all, it makes it easy to quickly assemble a basic web page without requiring any HTML knowledge.

However, it probably won't be long before you dip into the source view. Some types of changes are easier to make when you are working directly with your markup. Finding the control you want on the design surface, selecting it, and editing the properties one at a time in the Properties window can be tedious.

Notes

The source view is often helpful when you want to add plain HTML content after all, not everything in your web page needs to be a full-fledged web control. You can add ordinary HTML elements using design view (just drag the element you want from the HTML tab of the Toolbox), but it's often easier to type them in by hand, because you'll usually need to use a combination of elements to get the result you want.

2.3.1 The Web Form Markup

If you haven't written HTML pages before, the web page source might look a little intimidating. And if you have written HTML pages before, the web page source might look a little odd. That's because the source for an ASP.NET web form isn't 100 percent HTML. Instead, it's an HTML document with an extra ingredient ASP.NET web controls.

Every ASP.NET web form includes the standard HTML tags, like <html>, <head>, and <body>, which delineate the basic sections of your web page. You can insert additional HTML tags, like paragraphs of text (use the <p> tag), headings (use <h1>, <h2>, <h3>), tables (use <table>), and so on. Along with standard HTML, you can add ASP.NET-only elements to your web pages. For example, <asp:Button> represents a clickable button that triggers a task on the web server. When you add an ASP.NET web control, you create an object that you can interact with in your web page code, which is tremendously useful.

Here's a look at the web page shown in Figure 2.7. The details that are not part of ordinary HTML are highlighted, and the lines are numbered for easy reference:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form ID="form1" runat="server">
<div>
<asp:Label ID="Label1" runat="server"
Text="Type something here:" />
<asp:TextBox ID="TextBox1" runat="server" />
<br />
<asp:Button ID="Button1" runat="server" Text="Button" />
</div>
</form>
</body>
</html>
```

Obviously, the ASP.NET-specific details (the highlighted bits) don't mean anything to a web browser because they aren't valid HTML. This isn't a problem, because the web browser never sees these details. Instead, the ASP.NET engine creates an HTML "snapshot" of your page after all your code has finished processing on the server. At this point, details like the <asp:Button> are replaced with HTML tags that have the same appearance. The ASP.NET engine sends this HTML snapshot to the browser.

Notes

2.3.2 The Page Directive

The Default.aspx page, like all ASP.NET web forms, consists of three sections. The first section is the page directive:

```
<%@ Page Language="C#" AutoEventWireup="true"
       CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

The page directive gives ASP.NET basic information about how to compile the page. It indicates the language you're using for your code and the way you connect your event handlers. If you're using the code-behind approach (which is recommended), the page directive also indicates where the code file is located and the name of your custom page class. You won't need to modify the page directive by hand, because Visual Studio maintains it for you.

Notes The page directive is for ASP.NET's eyes only. The page directive doesn't appear in the HTML that's sent to the browser—instead, ASP.NET strips it out.

2.3.3 The Doctype

In an ordinary, non-ASP.NET web page, the doctype occupies the very first line. In an ASP.NET web form, the doctype gets second place, and appears just underneath the page directive.

The doctype indicates the type of markup (for example, HTML or XHTML) that you're using to create your web page. Technically, the doctype is optional, but Visual Studio adds it automatically. This is important, because depending on the type of markup you're using there may be certain tricks that aren't allowed. For example, strict XHTML doesn't let you use HTML formatting features that are considered obsolete and have been replaced by CSS.

The doctype is also important because it influences how a browser interprets your web page. For example, if you don't include a doctype on your web page, Internet Explorer (IE) switches itself into a legacy mode known as quirks mode. While IE is in quirks mode, certain formatting details are processed in inconsistent, nonstandard ways, simply because this is historically the way IE behaved. Later versions of IE don't attempt to change this behavior, even though it's faulty, because some websites may depend on it. However, you can specify a more standardized rendering that more closely matches the behavior of other browsers (like Firefox) by adding a doctype.

There are a small set of allowable doctypes that you can use. By default, newly created web pages in Visual Studio use the following doctype:

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

This indicates that the web page uses XHTML 1.0 transitional. The word transitional refers to the fact that this version of XHTML is designed to be a stepping stone between the old-fashioned HTML world and the ultra-strict XHTML world. XHTML transitional enforces all the structural rules of XHTML but allows some HTML formatting features that have been replaced by Cascading Style Sheets (CSS) and are considered obsolete.

If you don't need to use these details, you can step up to XHTML strict using this doctype:

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

These are the two most common doctypes in use today.

Notes

There are still a few more doctypes that you can use. If you're working with an existing website that's based on the somewhat out-of-date HTML standard, this is the doctype you need:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

And if you want to use the slightly tweaked XHTML 1.1 standard (rather than XHTML 1.0), you need the following doctype:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

XHTML 1.1 is mostly identical to XHTML 1.0 but streamlines a few more details and removes a few more legacy details. It doesn't provide a transitional option.

Notes There are a few more doctypes that you can use to create frames pages, which allow you to split a browser window into multiple panes, each of which shows a separate page. Frames pages are discouraged in modern day web development, because they don't work well with different window sizes and aren't always indexed correctly by search engines. You can see a more complete list of allowed doctypes, which includes the doctype for a frames page, at www.w3.org/QA/2002/04/Web-Quality.

Remember, the ASP.NET server controls will work equally well with any doctype. It's up to you to choose the level of standards compliance and backward compatibility you want in your web pages. If you're still in doubt, it's best to start out with XHTML 1.0 transitional, as it eliminates the quirks in different browser versions without removing all the legacy features. If you're ready to make a clean break with HTML, even if it means a bit more pain, consider switching to XHTML 1.0 strict or XHTML 1.1 (which is always strict) instead.

2.3.4 The Essentials of XHTML

Part of the goal of ASP.NET is to allow you to build rich web pages without forcing you to slog through the tedious details of XHTML (or HTML). ASP.NET delivers on this promise in many ways for example, in many situations you can get the result you want using a single slick web control rather than writing a page full of XHTML markup.

However, ASP.NET doesn't isolate you from XHTML altogether. In fact, a typical ASP.NET web page mingles ASP.NET web controls with ordinary XHTML content. When that page is processed by the web server, the ASP.NET web controls are converted to XHTML markup (a process known as rendering) and inserted into the page. The final result is a standard XHTML document that's sent back to the browser.

This design gives you the best of both worlds you can mix ordinary XHTML markup for the parts of your page that don't change, and use handy ASP.NET web controls for the parts that need to be interactive (such as buttons, lists, text boxes, and so on) or the parts that you need to update with new information (for example, a block of dynamic text). This design also suggests that ASP.NET developers should have a solid understanding of XHTML basics before they begin coding web forms. The following sections provide a brief overview that introduces you to the XHTML standard (or refreshes your memory, if you've learned it before). If you already know all you want to know about XHTML, feel free to skip ahead to the next section, "Writing Code."

Notes

Elements

The most important concept in the XHTML (and HTML) standard is the idea of elements. Elements are containers that contain bits of your web page content. For example, if you want to add a paragraph of text to a web page, you stuff it inside a paragraph element. A typical web page is actually composed of dozens (or hundreds) of elements. Taken together, these elements define the structure of the web page. They're also the starting point for formatting the web page. For example, headings usually look different than ordinary paragraphs, and tables look different than bulleted lists.

The XHTML language defines a small set of elements that you can use in fact, there are fewer than you probably expect. XHTML also defines the syntax for using these elements. A typical element consists of three pieces: a start tag, some content, and an end tag. Here's an example:

```
<p>This is a sentence in a paragraph.</p>
```

This example uses the paragraph element. The element starts with the start tag, ends with the end tag, and contains some text inside. Tags are easy to recognize, because they're always enclosed in angled brackets. And here's a combination that adds a heading to a web page followed by a paragraph:

```
<h1>A Heading</h1>
<p>This is a sentence in a paragraph.</p>
```

Browsers have built-in rules about how to process and display different elements. When a browser digests this markup, it always places the heading in a large, bold font, and adds a line break and some extra space underneath it, before starting the paragraph. Of course, there are ways to modify these formatting rules using the CSS standard.

Many XHTML elements can contain other elements. For example, you can use the <b> element inside the element to apply bold formatting to a portion of a paragraph:

```
<p>This is a <b>sentence</b> in a paragraph.</p>
```

The <h1> and elements usually hold content inside. As a result, they're split into a start tag and an end tag. For example, a heading begins with <h1> and ends with </h1>. However, some elements don't need any content, and can be declared using a special empty tag syntax that fuses the start and end tag together. For example, the <br> element represents a line break. Rather than writing
</br>, you can simply use <br />, as shown here:

```
<p>This is line one.<br />
This is line two.<br />
This is line three.</p>
```

Other elements that can be used in this fashion include <img> (for showing an image), <hr> (for creating a horizontal rule, or line), and most ASP.NET controls.

Table 2.2 lists some the most commonly used XHTML elements. The Type column distinguishes between two types of XHTML those that typically hold content or other nested elements (containers), and those that can be used on their own with the empty tag syntax you just considered (standalone).

Notes

Table 2.2: Basic XHTML Elements

Tag	Name	Type	Description
<b>, <i>, <u>	Bold, Italic, Underline	Container	These elements are used to apply basic formatting, and make text bold, italic, or underlined. Some web designers prefer to use instead of <b> and <emphasis> instead of <i>. Although these elements have the same standard rendering (bold and italic, respectively), they make more sense if you plan to use styles to change the formatting sometime in the future.
<p>	Paragraph	Container	The paragraph groups a block of freeflowing text together. The browser automatically adds a bit of space between paragraphs and other elements (like headings) or between subsequent paragraphs.
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>	Heading	Container	These elements are headings, which give text bold formatting and a large font size. The lower the number, the larger the text, so <h1> is for the largest heading. The <h5> heading is normal text size, and <h6> is actually a bit smaller than ordinary text.
<img>	Image	Standalone	The image element shows an external image file (specified by the src attribute) in a web page.
 	Line Break	Standalone	This element adds a single line break, with no extra space.
<hr>	Horizontal Line	Standalone	This element adds a horizontal line (which gets the full width of the containing element). You can use the horizontal line to separate different content regions.
<a>	Anchor	Container	The anchor element wraps a piece of text, and turns it into a link. You set the link target using the href attribute.
,	Unordered List, List Item	Container	These elements allow you to build bulleted lists. The element defines the list, while the element defines an item in the list (you nest the actual content for that item inside).
,	Ordered List, List Item	Container	These elements allow you to build numbered lists. The element defines the list, while the element defines an item in the list (you nest the actual content for that item inside).
<table>, <tr>, <td>	Table	Container	The <table> element allows you to create a multicolumn, multirow table. Each row is represented by a <tr> element inside the <table>. Each cell in a row is represented by a <td> element inside a <tr>. You place the actual content for the cell in the individual <td> elements.
	Division	Container	This element is an all-purpose container for other elements. It's used to separate different regions on the page, so you can format them or position them separately. For example, you can use a to create a shaded box around a group of elements.
	Span	Container	This element is an all-purpose container for bits of text content inside other elements (like headings or paragraphs). It's most commonly used to format those bits of text. For example, you can use a to change the color of a few words in a sentence.
<form>	Form	Container	This element is used to hold all the controls on a web page. Controls are HTML elements that can send information back to the web server when the page is submitted. For example, text boxes submit their text, list boxes submit the currently selected item in the list, and so on.

Notes

Task Specify the basic purpose of Doctype file in web application.

Attributes

Every XHTML document fuses together two types of information: the document content, and information about how that content should be presented. You control the presentation of your content in just three ways: by using the right elements, by arranging these elements to get the right structure, and by adding attributes to your elements.

Attributes are individual pieces of information that you attach to an element, inside the start tag. Attributes have many uses for example, they allow you to explicitly attach a style to an element so that it gets the right formatting. Some elements require attributes. The most obvious example is the <img> element, which allows you to pull the content from an image file and place it in your web page.

The <img> tag requires two pieces of information the image URL (the source), and the alternate text that describes the picture (which is used for accessibility purposes, as with screen reading software). These two pieces of information are specified using two attributes, named src and alt:

```
<img src="happy.gif" alt="Happy Face" />
```

The <a> anchor element is an example of an element that uses attributes and takes content.

The content inside the <a> element is the blue, underline text of the hyperlink. The href attribute defines the destination that the browser will navigate to when the link is clicked.

```
<p>
Click <a href="http://www.prosetech.com">here</a> to visit my website.
</p>
```

You'll use attributes extensively with ASP.NET control tags. With ASP.NET controls, every attribute maps to a property of the control class.

Formatting

Along with the <b> tag for bold, XHTML also supports <i> for italics and <u> for underlining. However, this is about as far its formatting goes.

XHTML elements are intended to indicate the structure of a document, not its formatting. Although you can adjust colors, fonts, and some formatting characteristics using XHTML elements, a better approach is to define formatting using a CSS style sheet. For example, a style sheet can tell the browser to use specific formatting for every <h1> element in a page. You can even apply the styles in a style sheet to all the pages in your website.

In an ASP.NET web page, there are two ways you can use CSS. You can use it directly to format elements. Or, you can configure the properties of the ASP.NET controls you're using, and they'll generate the styles they need automatically.

2.3.5 A Complete Web Page

You now know enough to put together a complete XHTML page.

Every XHTML document starts out with this basic structure (right after the doctype):

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

Notes

```
<title>Untitled Page</title>
</head>
<body>
</body>
</html>
```

When you create a new web form in Visual Studio, this is the structure you start with. Here's what you get:

1. XHTML documents start with the <html> tag and end with the </html> tag. This <html> element contains the complete content of the web page.
2. Inside the <html> element, the web page is divided into two portions. The first portion is the <head> element, which stores some information about the web page. You'll use this to store the title of your web page, which will appear in the title bar in your web browser. (You can also add other details here like search keywords, although these are mostly ignored by web browsers these days.) When you generate a web page in Visual Studio, the <head> section has a runat="server" attribute. This gives you the ability to manipulate it in your code.
3. The second portion is the <body> element, which contains the actual page content that appears in the web browser window.

In an ASP.NET web page, there's at least one more element. Inside the <body> element is a <form> element. The <form> element is required because it defines a portion of the page that can send information back to the web server. This becomes important when you start adding text boxes, lists, and other controls. As long as they're in a form, information like the current text in the text box and the current selection in the list will be sent to the web server using a process known as a postback. Fortunately, you don't need to worry about this detail yet just place all your web page content inside the <form> element.

Most of the time, when you're working with a page you'll focus on the markup inside the <form> tag, because that's the actual page content. When you create a new web page in Visual Studio, there's one more detail the element inside the <form> element:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form ID="form1" runat="server">
<div>
</div>
</form>
</body>
</html>
```

Strictly speaking, the element is optional it's just a container. You can think of it as an invisible box that has no built-in appearance or formatting. However, it's useful to use a tag to group portions of your page that you want to format in a similar way (for example, with the same font, background color, or border). That way, you can apply style settings to the tag, and they'll cascade down into every tag it contains. You can also create a real box on your page by giving the a border.

Notes

Notes The element is also useful because you can place text directly inside it, without needing a container element (such as a paragraph). On the other hand, adding text directly inside the <form> element violates the rules of XHTML.

Now you're ready to pop the rest of your content in the tag. If you add the Label and TextBox web controls, you'll end up with the same markup you created using the designer earlier in this unit but now you'll understand its markup underpinnings.

2.4 Writing Code

Many of Visual Studio's most welcome enhancements appear when you start to write the code that supports your user interface. To start coding, you need to switch to the code-behind view. To switch back and forth, you can use two View Code or View Designer buttons, which appear just above the Solution Explorer window. Another approach that works just as well is to doubleclick either the .aspx page in the Solution Explorer (to get to the designer) or the .aspx.cs page (to get to the code view). The "code" in question is the VB C# code, not the HTML markup in the .aspx file.

2.4.1 The Code-Behind Class

When you switch to code view, you'll see the page class for your web page. For example, if you've created a web page named SimplePage.aspx, you'll see a code-behind class that looks like this:

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
public partial class SimplePage: System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
}
}
```

Just before your page class, Visual Studio imports a number of core .NET namespaces. These namespaces give you easy access to many commonly used ASP.NET classes.

Inside your page class you can place methods, which will respond to control events. For example, you can add a method with code that reacts when the user clicks a button. The following section explains how you can create an event handler.

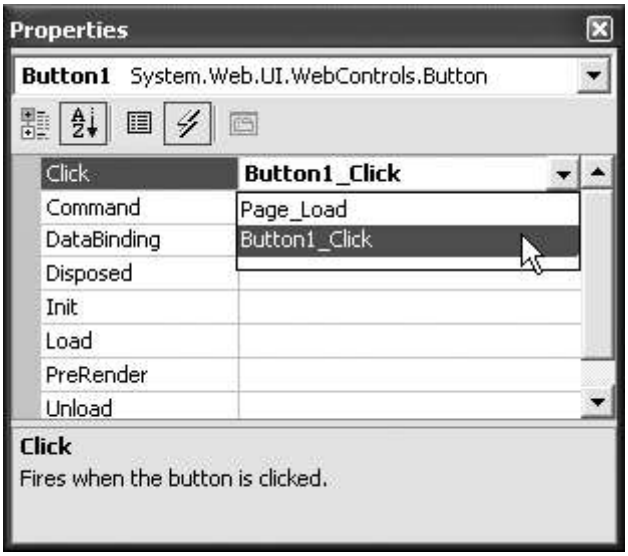
Notes

2.4.2 Adding Event Handlers

Most of the code in an ASP.NET web page is placed inside event handlers that react to web control events. Using Visual Studio, you have three easy ways to add an event handler to your code:

1. ***Type it in Manually:*** In this case, you add the subroutine directly to the page class. You must specify the appropriate parameters.
2. ***Double-click a Control in Design view:*** In this case, Visual Studio will create an event handler for that control's default event, if it doesn't already exist. For example, if you double-click a Button control, it will create an event handler for the Button.Click event. If you double-click a TextBox control, you'll get an event handler for the TextBox.TextChanged event.
3. ***Choose the Event from the Properties Window:*** Just select the control, and click the lightning bolt in the Properties window. You'll see a list of all the events provided by that control. Double-click next to the event you want to handle, and Visual Studio will automatically generate the event handler in your page class. Alternatively, if you've already created the event handler method, just select the event in the Properties window, and click the drop-down arrow at the right. You'll see a list that includes all the methods in your class that match the signature this event requires. You can then choose a method from the list to connect it. Figure 2.11 shows an example where the Button.Click event is connected to the Button1_Click method in the page class.

Figure 2.11: Creating or Attaching an event Handler



No matter which approach you use, the event handler looks (and functions) the same.

Example: When you double-click a Button control, Visual Studio creates an event handler like this:

```
protected void Button1_Click(object sender, EventArgs e)
{
// Your code for reacting to the button click goes here.
}
```

Notes

When you use Visual Studio to attach or create an event handler, it adjusts the control tag so that it's linked to the appropriate event:

```
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click"
/>
```

Inside your event handler method, you can interact with any of the control objects on your web page using their IDs. For example, if you've created a TextBox control named TextBox1, you can set the text using the following line of code:

```
protected void Button1_Click(object sender, EventArgs e)
{
TextBox1.Text = "Here is some sample text.";
}
```

This is a simple event handler that reacts when Button1 is clicked and updates the text in TextBox1.

Notes You might wonder why your code file includes the event handlers, but it doesn't actually declare the controls that you use (like the Button1 and TextBox1 objects in the previous example). The reason is that ASP.NET generates the declarations for these controls automatically. You'll never see these declarations, but you can assume they're a part of your class. That's also why every page class you create is defined with the partial keyword. This allows ASP.NET to merge your code with the portion it generates automatically. The end result is that you can easily access all the controls on your page by name, but you don't need to bother with extra code to create and initialize these objects.

2.4.3 IntelliSense and Outlining

Visual Studio provides a number of automatic time-savers through its IntelliSense technology. They are similar to features such as automatic spell checking and formatting in Microsoft Office applications. This unit introduces most of these features, but you'll need many hours of programming before you'll become familiar with all of Visual Studio's time-savers. We don't have enough space to describe advanced tricks such as the intelligent search-and replace features and Visual Studio's programmable macros. These features could occupy an entire book of their own!

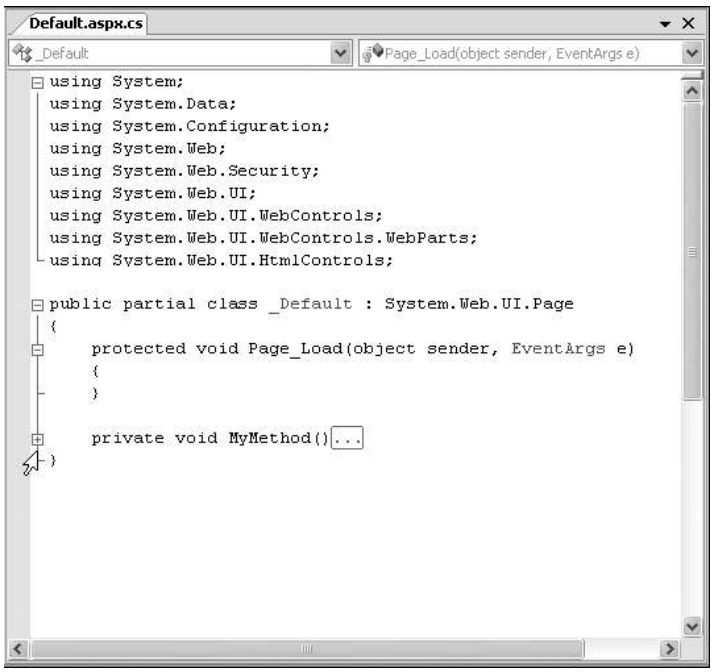
Outlining

Outlining allows Visual Studio to "collapse" a method, class, structure, namespace, or region to a single line. It allows you to see the code that interests you while hiding unimportant code. To collapse a portion of code, click the minus (–) symbol next to the first line. To expand it, click the box again, which will now have a plus (+) symbol (see Figure 2.12).

You can hide every method at once by right-clicking anywhere in the code window and choosing Outlining → Collapse to Definitions.

Notes

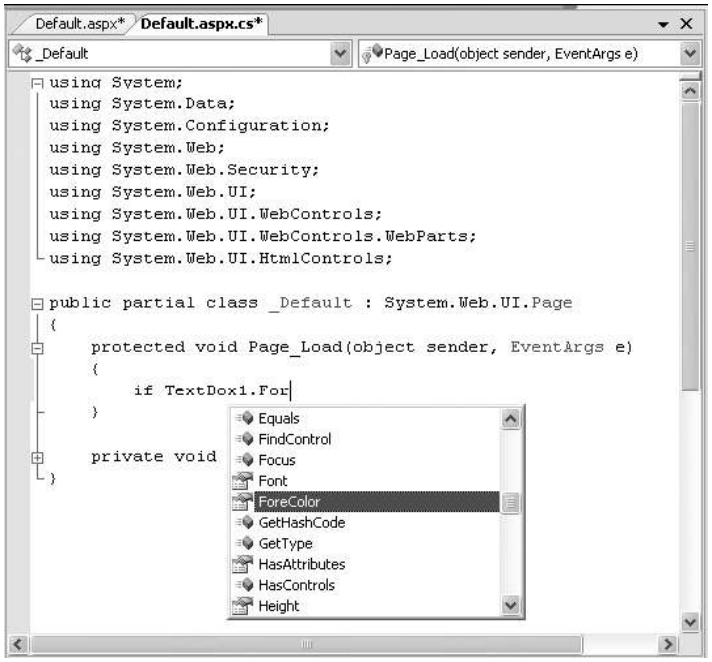
Figure 2.12: Collapsing Code



Member List

Visual Studio makes it easy for you to interact with controls and classes. When you type a class or object name, it pops up a list of available properties and methods (Figure 2.13). It uses a similar trick to provide a list of data types when you define a variable or to provide a list of valid values when you assign a value to an enumeration.

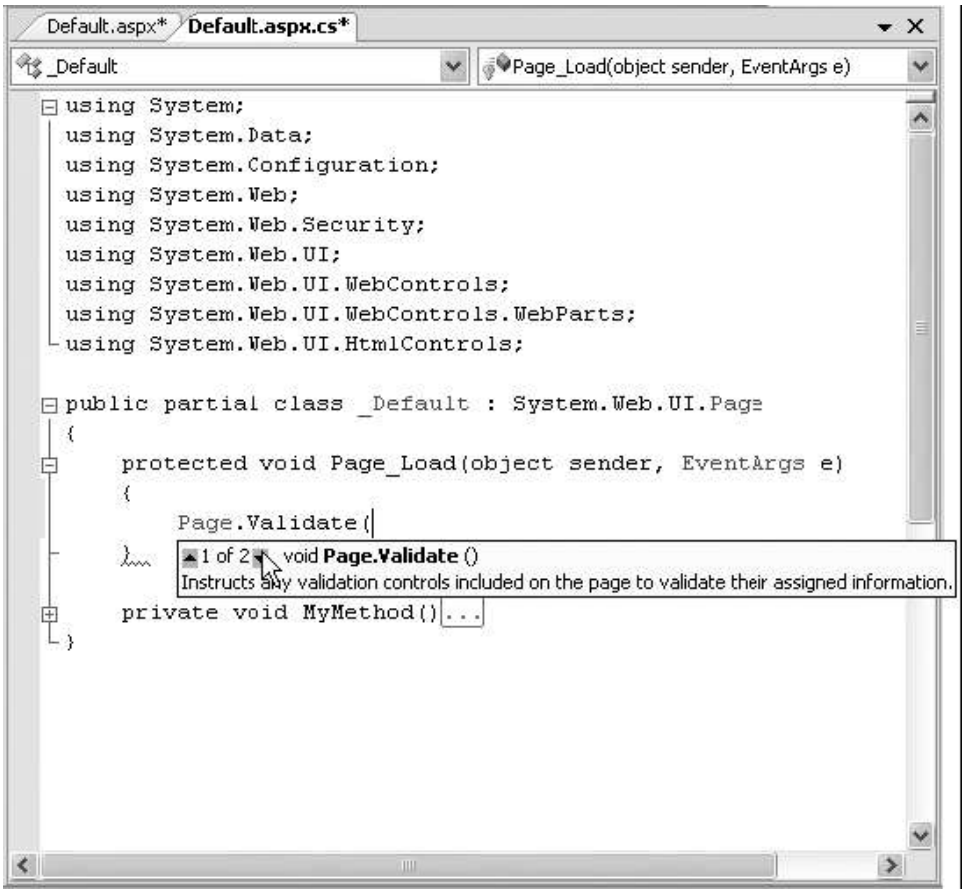
Figure 2.13: IntelliSense at Work



Notes

Visual Studio also provides a list of parameters and their data types when you call a method or invoke a constructor. This information is presented in a tooltip below the code and appears as you type. Because the .NET class library uses method overloading a lot, these methods may have multiple versions. When they do, Visual Studio indicates the number of versions and allows you to see the method definitions for each one by clicking the small up and down arrows in the tooltip. Each time you click the arrow, the tooltip displays a different version of the overloaded method (Figure 2.14).

Figure 2.14: IntelliSense with Overloaded Methods

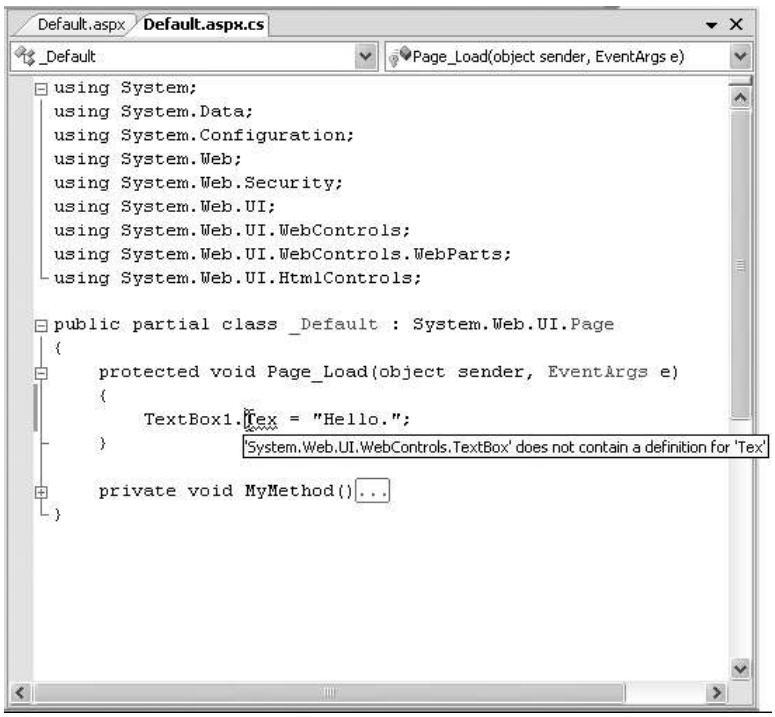


Error Underlining

One of the code editor's most useful features is error underlining. Visual Studio is able to detect a variety of error conditions, such as undefined variables, properties, or methods; invalid data type conversions; and missing code elements. Rather than stopping you to alert you that a problem exists, the Visual Studio editor underlines the offending code. You can hover your mouse over an underlined error to see a brief tooltip description of the problem (Figure 2.15).

Notes

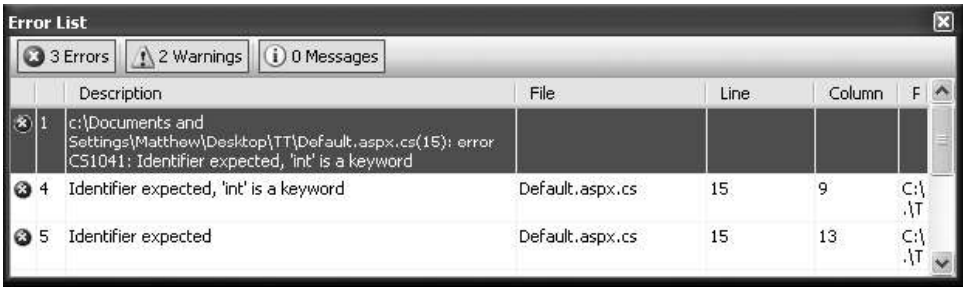
Figure 2.15: Highlighting Errors at Design Time



Visual Studio won't necessarily flag your errors immediately. But when you try to run your application (or just compile it), Visual Studio will quickly scan through the code, marking all the errors it finds. If your code contains at least one error, Visual Studio will ask you whether it should continue. At this point, you'll almost always decide to cancel the operation and fix the problems Visual Studio has discovered. (If you choose to continue, you'll actually wind up using the last compiled version of your application, because Visual Studio can't build an application that has errors.)

Whenever you attempt to build an application that has errors, Visual Studio will display the Error List window with a list of all the problems it detected, as shown in Figure 2.16. You can then jump quickly to a problem by double-clicking it in the list.

Figure 2.16: The Error List Window



You may find that as you fix errors and rebuild your project, you discover more problems. That's because Visual Studio doesn't check for all types of errors at once. When you try to compile your application, Visual Studio scans for basic problems such as unrecognized class names. If these problems exist, they can easily mask other errors. On the other hand, if your code

Notes

passes this basic level of inspection, Visual Studio checks for more subtle problems such as trying to use an unassigned variable.

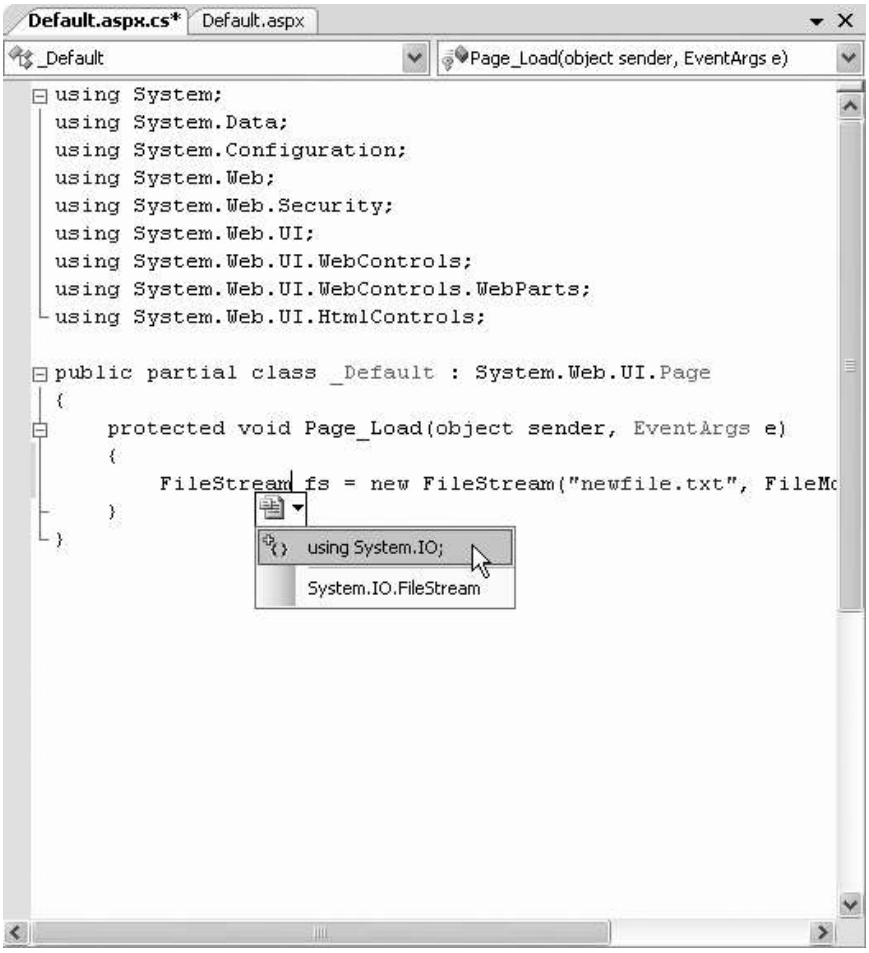
You can also configure the level of error checking Visual Studio performs for markup in your .aspx files. Usually, you'll want to set the level of validation to match the doctype that you're using. Unfortunately, Visual Studio doesn't take this step automatically. Instead, it's up to you to choose the level of validation you want from the drop-down list in the HTML Source Editing toolbar. (If the HTML Source Editing toolbar is not currently displayed, right-click the toolbar strip and choose HTML Source Editing.) The most common validation choices are HTML 4.01, XHTML 1.0 Transitional, and XHTML 1.1. For example, if you choose XHTML 1.0 Transitional or XHTML 1.1, you'll receive a warning in the Error List if your web page includes syntax that's not legal in XHTML, like incorrect capitalization, an obsolete formatting attribute, or an element that's not properly closed. You'll still be able to run your web page, but you'll know that your page isn't completely consistent with the XHTML standard.

Automatically Importing Namespaces

Sometimes, you'll run into an error because you haven't imported a namespace that you need. For example, imagine you type a line of code like this:

```
FileStream fs = new FileStream("newfile.txt", FileMode.Create);
```

Figure 2.17: Build Errors in the Error List



Notes

This line creates an instance of the FileStream class, which resides in the System.IO namespace. However, if you haven't imported the System.IO namespace, you'll run into a compile-time error. Unfortunately, the error simply indicates no known class named FileStream exists—it doesn't indicate whether the problem is a misspelling or a missing import, and it doesn't tell you which namespace has the class you need.

Visual Studio offers an invaluable tool to help you in this situation. When you move the text cursor to the unrecognized class name (FileStream in this example), a small box icon appears underneath. If you hover over that location with the mouse, a page icon appears. Click the page icon, and a drop-down list of autocorrect options appear (Figure 2.17). Using these options, you can convert the line to use the fully qualified class name or add the required namespace import to the top of your code file, which is generally the cleanest option (particularly if you use classes from that namespace more than once in the same page).

The only case when this autocorrect feature won't work is if Visual Studio can't find the missing class. This might happen if the class exists in another assembly, and you haven't added a reference to that assembly yet.

Auto Format and Color

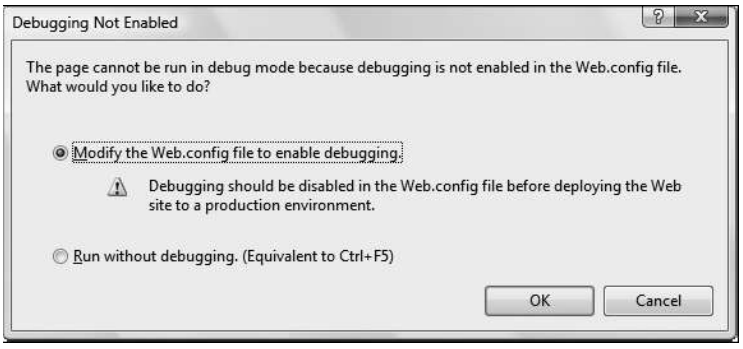
Visual Studio also provides some cosmetic conveniences. It automatically colors your code, making comments green, keywords blue, and normal code black. The result is much more readable code. You can even configure the colors Visual Studio uses by selecting Tools → Options and then choosing the Environment → Fonts and Colors section.

In addition, Visual Studio is configured by default to automatically format your code. This means you can type your code lines freely without worrying about tabs and positioning. Visual Studio automatically applies the "correct" indenting. Fortunately, if you have a different preference (for example, you want five spaces instead of four spaces of indenting, or you want to use tabs instead of spaces), you can configure this behavior. Just select Tools → Options, and find the Text Editor → C# group of settings.

2.5 Visual Studio Debugging

Once you've created an application, you can compile and run it by choosing Debug ä Start Debugging from the menu or by clicking the Start Debugging button on the toolbar (which looks like a DVD-style play button). Visual Studio launches your default web browser and requests the page that's currently selected in the Solution Explorer. This is a handy trick if you're in the middle of coding SalesPage1.aspx, you'll see SalesPage1.aspx appear in the browser, not the Default.aspx home page.

Figure 2.18: Enabling Debugging



Notes

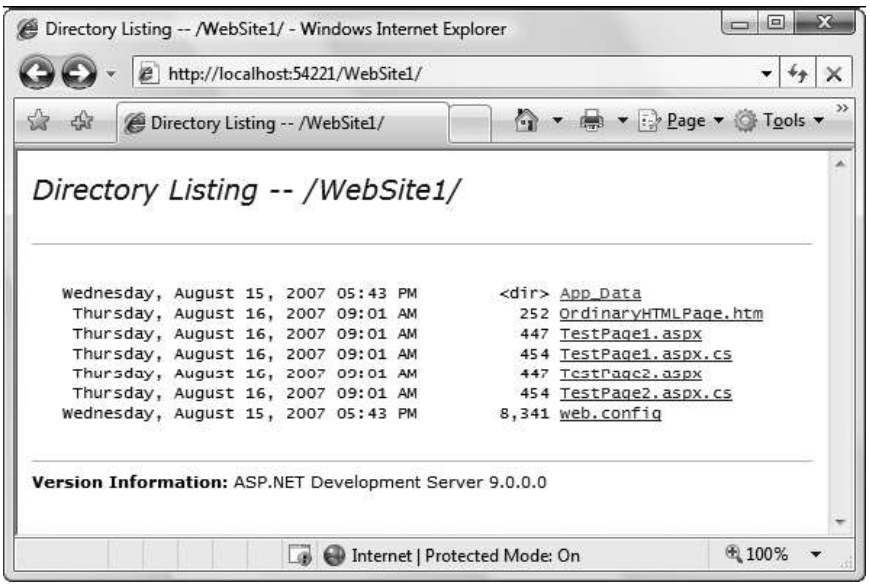
The first time you launch a web application, Visual Studio will ask you whether you want to configure your web application to allow debugging by adjusting its configuration file. (Figure 2.18 shows the message you'll see.) Choose "Modify the Web.config file to enable debugging." and click OK.

Visual Studio may also warn you that script debugging is disabled, depending on your browser preferences. Script debugging is a useful tool that works with Visual Studio to help you debug pages that use ASP.NET AJAX. However, there's no reason to turn script debugging on unless you're writing client-side JavaScript code in your web pages. (By default, script debugging is disabled in Internet Explorer so that you don't get error messages when you run someone else's problematic JavaScript code when visiting a website.) It's a good idea to choose the "Don't show this dialog again" to make sure Visual Studio doesn't repeat the same warning every time you run your web application.

2.5.1 The Visual Studio Web Server

When you run a web application, Visual Studio starts its integrated web server. Behind the scenes, ASP.NET compiles the code for your web application, runs your web page, and then returns the final HTML to the browser. The first time you run a web page, you'll see a new icon appear in the system tray at the bottom-right corner of the taskbar. This icon is Visual Studio's test web server, which runs in the background hosting your website. The test server only runs while Visual Studio is running, and it only accepts requests from your computer (so other users can't connect to it over a network).

Figure 2.19: Choosing from a List of Pages



When you run a web page, you'll notice that the URL in the browser includes a port number. For example, if you run a web application in a folder named OnlineBank, you might see a URL like http://localhost:4235/OnlineBank/Default.aspx. This URL indicates that the web server is running on your computer (localhost), so its requests aren't being sent over the Internet. It also indicates that all requests are being transmitted to port number 4235. That way, the requests won't conflict with any other applications that might be running on your computer and listening for requests. Every time Visual Studio starts the integrated web server, it randomly chooses an available port.

Notes

Visual Studio's built-in web server also allows you to retrieve a listing of all the files in your website. This means if you create a web application named SampleSite, you can request it in the form http://localhost:port/SampleSite (omitting the page name) to see a list of all the files in your web application folder (Figure 2.19). Then, just click the page you want to test.

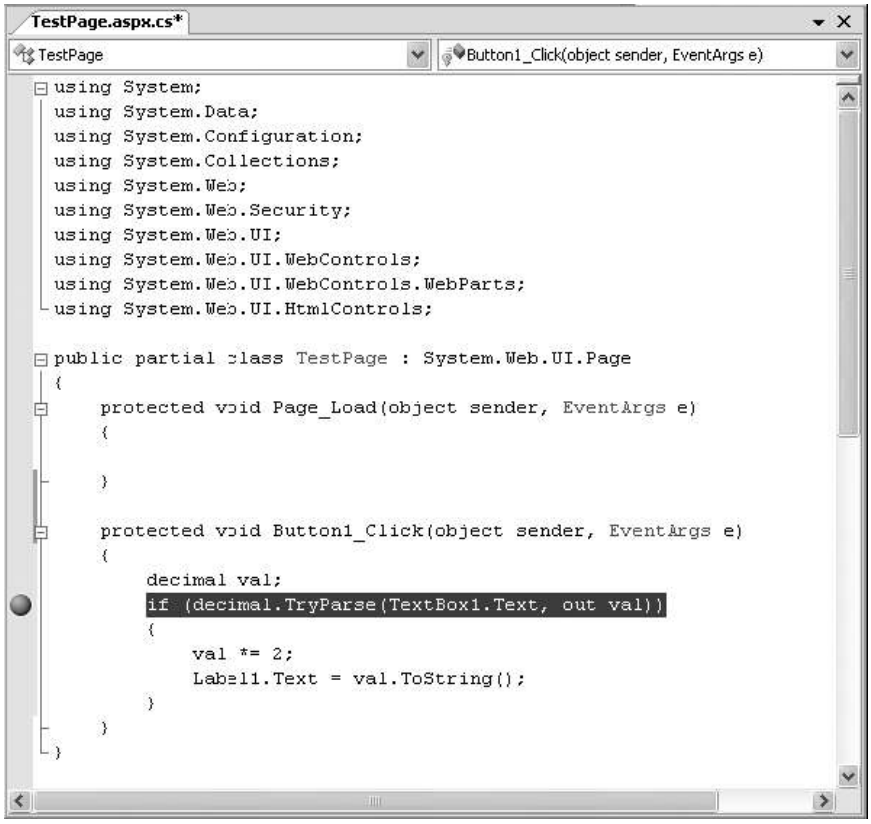
This trick won't work if you have a Default.aspx page. If you do, any requests that don't indicate the page you want are automatically redirected to this page.

2.5.2 Single-step Debugging

Single-step debugging allows you to test your assumptions about how your code works and see what's really happening under the hood of your application. It's incredibly easy to use. Just follow these steps:

1. Find a location in your code where you want to pause execution. (You can use any executable line of code but not a variable declaration, comment, or blank line.) Click in the margin next to the line of code, and a red breakpoint will appear (Figure 2.20).

Figure 2.20: Setting a Breakpoint



2. Now start your program as you would ordinarily (by pressing the F5 key or using the Start button on the toolbar). When the program reaches your breakpoint, execution will pause, and you'll be switched to the Visual Studio code window. The breakpoint statement won't be executed yet.

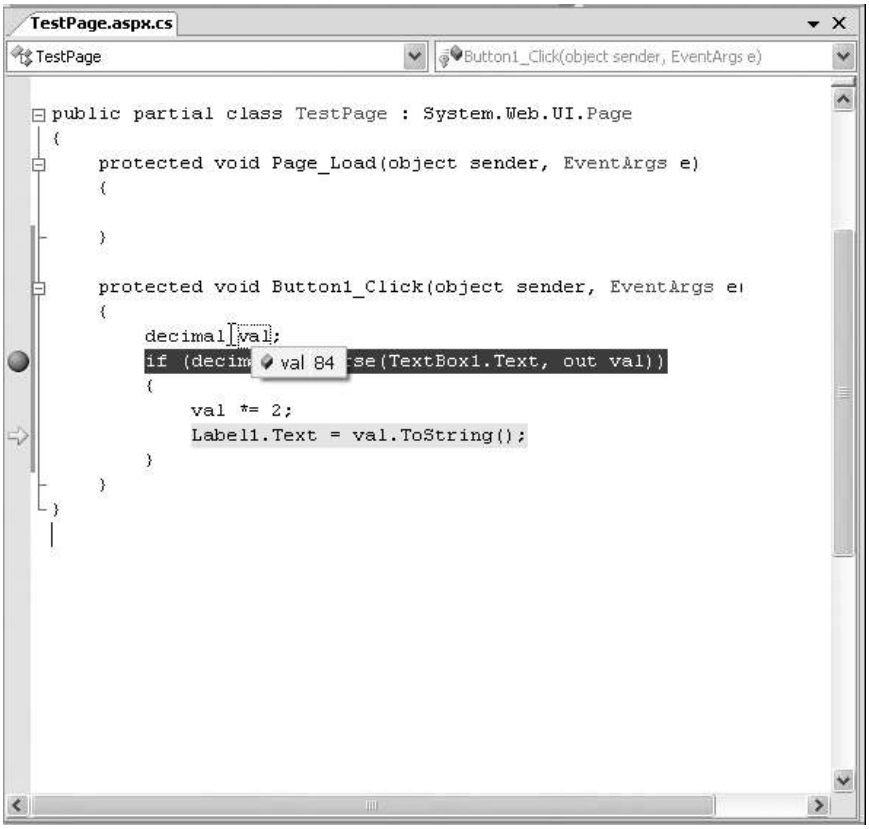
3. At this point, you have several options. You can execute the current line by pressing F11. The following line in your code will be highlighted with a yellow arrow, indicating that

Notes

this is the next line that will be executed. You can continue like this through your program, running one line at a time by pressing F11 and following the code's path of execution.

4. Whenever the code is in break mode, you can hover over variables to see their current contents (Figure 2.21). This allows you to verify that variables contain the values you expect.

Figure 2.21: Viewing Variable Contents in Break Mode



5. You can also use any of the commands listed in Table 2.3 while in break mode. These commands are available from the context menu by right-clicking the code window or by using the associated hot key.

Table 2.3: Commands Available in Break Mode

Command (Hot Key)	Description
Step Into (F11)	Executes the currently highlighted line and then pauses. If the currently highlighted line calls a method, execution will pause at the first executable line inside the method (which is why this feature is called stepping into).
Step Over (F10)	The same as Step Into, except it runs methods as though they are a single line. If you select Step Over while a method call is highlighted, the entire method will be executed. Execution will pause at the next executable statement in the current method.
Step Out (Shift+F11)	Executes all the code in the current procedure and then pauses at the statement that immediately follows the one that called this method or function. In other words, this allows you to step "out" of the current procedure in one large jump.

Contd...

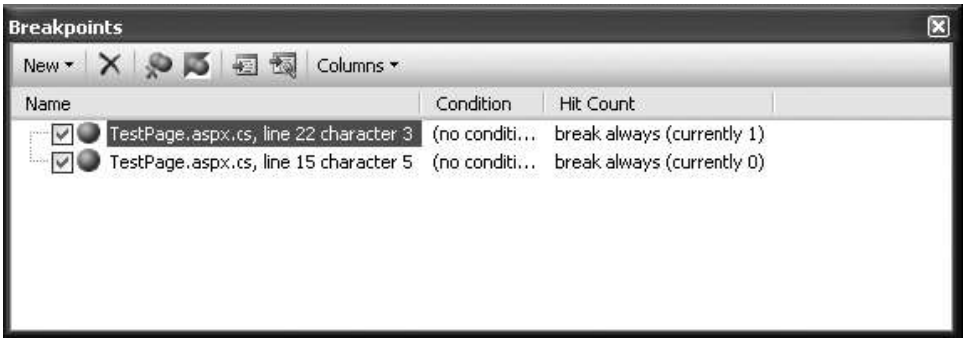
Notes

Continue (F5)	Resumes the program and continues to run it normally, without pausing until another breakpoint is reached.
Run to Cursor	Allows you to run all the code up to a specific line (where your cursor is currently positioned). You can use this technique to skip a time consuming loop.
Set Next Statement	Allows you to change the path of execution of your program while debugging. This command causes your program to mark the current line (where your cursor is positioned) as the current line for execution. When you resume execution, this line will be executed, and the program will continue from that point. Although this technique is convenient for jumping over large loops and simulating certain conditions, it's easy to cause confusion and runtime errors by using it recklessly.
Show Next Statement	Brings you to the line of code where Visual Studio is currently halted. (This is the line of code that will be executed next when you continue.) This line is marked by a yellow arrow. The Show Next Statement command is useful if you lose your place while editing.

You can switch your program into break mode at any point by clicking the Pause button in the toolbar or selecting Debug → Break All. This might not stop your code where you expect, however, so you'll need to rummage around to get your bearings.

When debugging a large website, you might place breakpoints in different places in your code and in multiple web pages. To get an at-a-glance look at all the breakpoints in your web application, choose Debug → Windows → Breakpoints. You'll see a list of all your breakpoints, as shown in Figure 2.22.

Figure 2.22: The Breakpoints Window



You can jump to the location in code where a breakpoint is placed by double-clicking it in the list. You can also remove a breakpoint (select it and press Delete) or temporarily disable a breakpoint (by removing the check mark next to it). This allows you to keep a breakpoint to use in testing later, without leaving it active.

2.5.3 Variable Watches

In some cases, you might want to track the status of a variable without switching into break mode repeatedly. In this case, it's more useful to use the Autos, Locals, and Watch windows, which allow you to track variables across an entire application. Table 2.4 describes these windows.

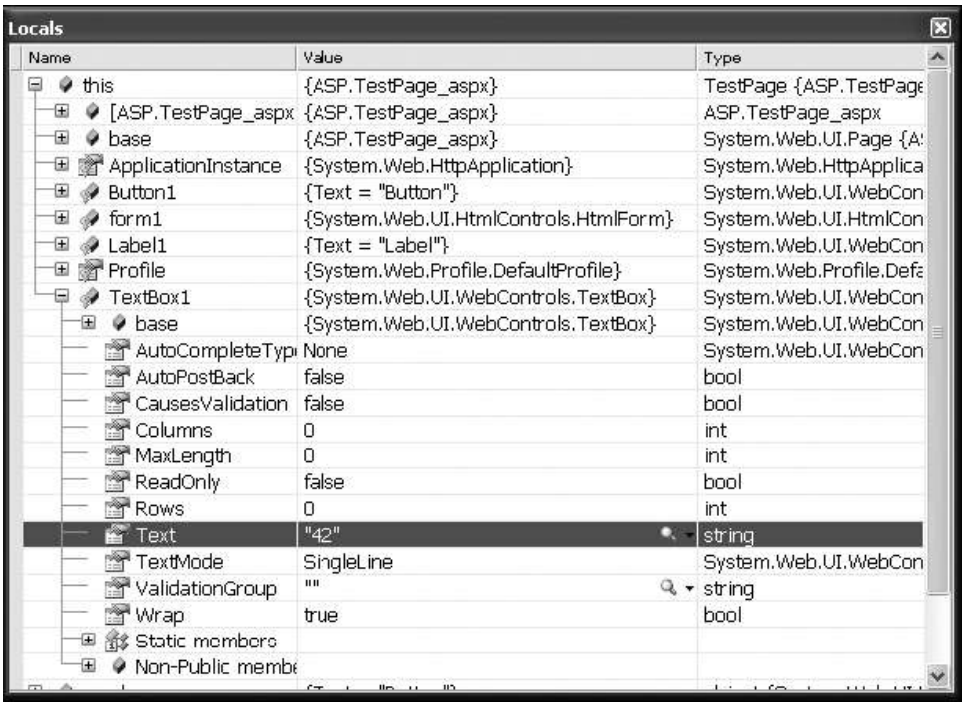
Notes

Table 2.4: Variable Watch Windows

Window	Description
Autos	Window Automatically displays variables that Visual Studio determines are important for the current code statement. For example, this might include variables that are accessed or changed in the previous line.
Locals	Automatically displays all the variables that are in scope in the current method. This offers a quick summary of important variables.
Watch	Displays variables you have added. Watches are saved with your project, so you can continue tracking a variable later. To add a watch, right-click a variable in your code, and select Add Watch; alternatively, double-click the last row in the Watch window, and type in the variable name.

Each row in the Autos, Locals, and Watch windows provides information about the type or class of the variable and its current value. If the variable holds an object instance, you can expand the variable and see its members and properties. For example, in the Locals window you'll see the variable this (Figure 2.23), which is a reference to the current object inside of which your code is executing (in this case, the web page). If you click the plus (+) sign next to the word this, a full list will appear that describes many page properties (and some system values).

Figure 2.23: Viewing the Current Page Object in the Locals Window



If you are missing one of the Watch windows, you can show it manually by selecting it from the Debug → Windows submenu.

Notes

Case Study

Exploring ASP.NET with the IBuySpy

The IBuySpy Insight: IBuySpy is similar to other Microsoft case studies (like Duwamish and Fitch & Mather Stocks), but because it was created from the ground up for ASP.NET, it's an excellent place to start learning good ASP.NET application design. The most surprising revelation for experienced developers will probably be the simplicity of the project. For example, here are a few things you won't see in the IBuySpy examples:

1. ***COM+ Serviced Components.*** The connection pooling features of ADO.NET mean that you may not need to use COM+ in a distributed application. ADO.NET automatically creates and retains a pool of database connections that can be handed out to any ASP.NET worker thread in need, avoiding some of the traditional overhead of a database call.
2. ***Business Objects.*** Instead, IBuySpy uses an optimized design where pages talk to the database through a set of dedicated database classes. However, these classes (which we'll explore a little later) don't shoulder any responsibility for enforcing so-called "business rules," which generally need to be handled further upstream.
3. ***Custom validation, authentication, or caching code.*** Workarounds were common for all three of these in typical ASP sites. In ASP.NET, platform services take care of all the heavy lifting.

So what are some of the key insights you can glean from playing with IBuySpy? Here are some examples:

1. ***User controls organize UI functionality.*** For example, menu navigational controls are encapsulated in dedicated controls, which are then repeated on multiple pages. More complex inherited controls, which are primarily of interest to tool vendors, aren't used.
2. ***Output caching is used to avoid the database.*** For example, the IBuySpy e-commerce site caches the output for the product category navigation control for 100 minutes. This means that changes to the product categories won't be picked up immediately. However, it also ensures that thousands of e-shoppers can shop at a time, without requiring more than one database trip. Data caching, which allows programmers to implement more flexible caching strategies, is avoided — in fact, it isn't needed.
3. ***Session state is avoided.*** Instead, per-user information (like the list of items in the shopping cart) is stored in the database. This gives the developer the freedom to retain shopping cart items for long periods of time, and not worry about wasting valuable server memory.

Perhaps the most interesting aspect is the strict tiered design of the IBuySpy case studies. The ASP.NET Web page code never talks directly to the database. Instead, the Web page code talks to a layer of database classes. A separate source code file is included for every table the application needs to use in the database. This file includes an entity class and a stateless service provider class.

2.6 Summary

- In this unit, you took a quick look at Visual Studio 2008. First, you saw how to create a new web application using the clean projectless website model.

Notes

- Next, you considered how to design basic web pages, complete with controls and code.
- Finally, you saw how to use Visual Studio's rich set of debugging features to get into the brain of your web page code and track down elusive problems.

2.7 Keywords

Visual Basic .NET (VB.NET): A Microsoft-supported language for the .NET Framework.

Visual Studio .NET (VS .NET): A full-featured, Interactive Development Environment (IDE) created by Microsoft for the development of .NET applications. VS .NET makes a better alternative to Visual Notepad for creating .NET applications.

Web Form: A .NET Framework object that allows development of Web-based applications and Web sites.

Web Service: An application hosted on a Web server that provides information and services to other network applications using the HTTP and XML protocols. A Web service is conceptually an URL-addressable library of functionality that is completely independent of the consumer and stateless in its operation.

Web Services Description Language Tool: A .NET programming tool (Wsdl.exe) used to create service descriptions and generate proxies for ASP.NET Web service methods.

Windows Form: A .NET Framework object that allows the development of "traditional" Windows desktop applications.

2.8 Self Assessment

Fill in the blanks:

1. In ..... a web application is called a web site.
2. Visual Studio allows you to create .....applications without project files.
3. To configure a control in design view you must first select it on the .....
4. Every ASP.NET web form includes the standard HTML tags, like .....
5. The ..... indicates the type of markup that you're using to create your web page.
6. ..... are individual pieces of information that you attach to an element, inside the start tag.

State whether the following statements are True or False:

7. In an ASP.NET web page, there are two ways you can use CSS.
8. Visual Studio provides a number of automatic time-savers through its IntelliSense technology.
9. Outlining does not allow Visual Studio to "collapse" a method, class, structure, namespace, or region to a single line.
10. Script debugging is a useful tool that works with Visual Studio to help you debug pages that use ASP.NET AJAX.

Notes

2.9 Review Questions

1. How do you create a new web site in the IDE?
2. What are the three views of your page that you can use in the IDE?
3. What's the name for the settings that are specific to each control?
4. Where in the IDE will you find the controls that you can place on your page?
5. How do you run your application?
6. What event is fired when you click on the Button control?
7. Where is the code for the event handler located?
8. What's one way to access the default event handler's code?
9. What property of the Label control do you use to set its content?
10. When you click the button in your Hello World application, where is the code processed?

Answers: Self Assessment

1.	Visual Studio 2008	2.	ASP.NET
3.	page	4.	<html>, <head>, and <body>
5.	doctype	6.	Attributes
7.	True	8.	True
9.	False	10.	True

2.10 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB.*, Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Unit 3: Web Form Fundamentals

Notes

CONTENTS

Objectives

Introduction

3.1 ASP.NET Application

3.1.1 ASP.NET File Types

3.1.2 ASP.NET Application Directories

3.2 Introducing Server Controls

3.3 The Html Control Classes

3.3.1 HTML Control Events

3.3.2 Advanced Events with the HtmlInputImage Control

3.3.3 The HtmlControl Base Class

3.3.4 The HtmlContainerControl Class

3.3.5 The HtmlInputControl Class

3.4 The Page Class

3.5 Application Events

3.5.1 The Global.asax File

3.5.2 Additional Application Events

3.6 ASP.NET Configuration

3.6.1 The web.config File

3.6.2 Nested Configuration

3.6.3 Storing Custom Settings in the web.config File

3.6.4 The Website Administration Tool (WAT)

3.7 Summary

3.8 Keywords

3.9 Self Assessment

3.10 Review Questions

3.11 Further Readings

Notes

Objectives

After studying this unit, you will be able to:

- Know ASP.NET application
- Define server controls
- Describe HTML controls classes
- Explain application events
- Know ASP.NET configuration

Introduction

ASP.NET evolved from Microsoft's earlier Web technology called Active Server Pages (referred to as ASP then and classic ASP today). This model was completely different from today's ASP.NET. Classic ASP used interpreted languages to accomplish the construction of the final HTML document before it was sent to the browser. ASP.NET, on the other hand, uses true compiled languages to accomplish the same task. The idea of building Web pages based on objects in a compiled environment is one of the main focuses of this unit .

This unit looks at how to use a particular type of object in ASP.NET pages called a server control, and how you can profit from using this control. We also introduce a particular type of server control the HTML server control. The unit also demonstrates how you can use JavaScript in ASP.NET pages to modify the behavior of server controls.

3.1 ASP.NET Application

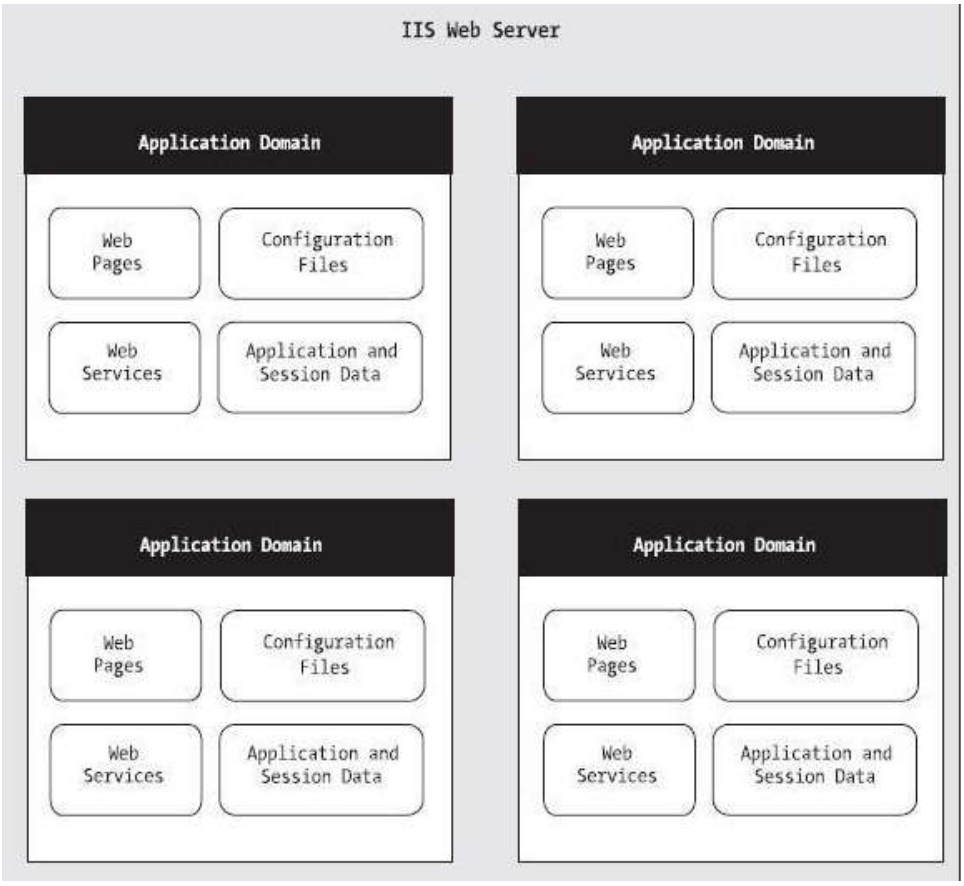
It's sometimes difficult to define exactly what a web application is. Unlike a traditional desktop program (which users start by running a stand-alone EXE file), ASP.NET applications are almost always divided into multiple web pages. This division means a user can enter an ASP.NET application at several different points or follow a link from the application to another part of the website or another web server. So, does it make sense to consider a website as an application?

In ASP.NET, the answer is yes. Every ASP.NET application shares a common set of resources and configuration settings. Web pages from other ASP.NET applications don't share these resources, even if they're on the same web server. Technically speaking, every ASP.NET application is executed inside a separate application domain. Application domains are isolated areas in memory, and they ensure that even if one web application causes a fatal error, it's unlikely to affect any other application that is currently running on the same computer. Similarly, application domains restrict a web page in one application from accessing the in memory information of another application. Each web application is maintained separately and has its own set of cached, application, and session data.

The standard definition of an ASP.NET application describes it as a combination of files, pages, handlers, modules, and executable code that can be invoked from a virtual directory (and, optionally, its subdirectories) on a web server. In other words, the virtual directory is the basic grouping structure that delimits an application. Figure 3.1 shows a web server that hosts four separate web applications.

Notes

Figure 3.1: ASP.NET Applications



3.1.1 ASP.NET File Types

ASP.NET applications can include many types of files. Table 3.1 introduces the essential ingredients.

Table 3.1: ASP.NET File Types

File Name	Description
Ends with .aspx	These are ASP.NET web pages (the .NET equivalent of the .asp file in an ASP application). They contain the user interface and, optionally, the underlying application code. Users request or navigate directly to one of these pages to start your web application.
Ends with .ascx	These are ASP.NET user controls. User controls are similar to web pages, except that the user can't access these files directly. Instead, they must be hosted inside an ASP.NET web page. User controls allow you to develop a small piece of user interface and reuse it in as many web forms as you want without repetitive code.
Ends with .asmx	These are ASP.NET web services collections of methods that can be called over the Internet. Web services work differently than web pages, but they still share the same application resources, configuration settings, and memory.

Contd...

Notes

web.config	This is the XML-based configuration file for your ASP.NET application. It includes settings for customizing security, state management, memory management, and much more.
Global.asax	This is the global application file. You can use this file to define global variables (variables that can be accessed from any web page in the web application) and react to global events (such as when a web application first starts).
Ends with .cs	These are code-behind files that contain C# code. They allow you to separate the application logic from the user interface of a web page.

In addition, your web application can contain other resources that aren't special ASP.NET files. For example, your virtual directory can hold image files, HTML files, or CSS files. These resources might be used in one of your ASP.NET web pages, or they might be used independently. A website could even combine static HTML pages with dynamic ASP.NET pages.

Most of the file types in Table 3.1 are optional. You can create a legitimate ASP.NET application with a single .aspx web page file.

3.1.2 ASP.NET Application Directories

Every web application should have a well-planned directory structure. For example, you'll probably want to store images in a separate folder from where you store your web pages. Or you might want to put public ASP.NET pages in one folder and restricted ones in another so you can apply different security settings based on the directory.

Along with the directories you create, ASP.NET also uses a few specialized subdirectories, which it recognizes by name (Table 3.2). Keep in mind that you won't see all these directories in a typical application. Visual Studio will prompt you to create them as needed.

Table 3.2: ASP.NET Directories

Directory	Description
Bin	Contains all the compiled .NET components (DLLs) that the ASP.NET web application uses. For example, if you develop a custom component for accessing a database, you'll place the component here. ASP.NET will automatically detect the assembly, and any page in the web application will be able to use it. This seamless deployment model is far easier than working with traditional COM components, which must be registered before they can be used (and often reregistered when they change).
App_Code	Contains source code files that are dynamically compiled for use in your application. You can use this directory in a similar way as the Bin directory; the only difference is that you place source code files here instead of compiled assemblies.
App_Global Resources	This directory is used in localization scenarios, when you need to have a website in more than one language.
App_LocalRe sources	Serves the same purpose as App_GlobalResources, except these resources are accessible to a specific page only.
App_WebRef erences	Stores references to web services that the web application uses.
App_Data	Stores data, including SQL Server 2005 Express Edition database files and XML files. Of course, you're free to store data files in other directories.
App_Themes	Stores the themes that are used by your web application.

Notes

3.2 Introducing Server Controls

ASP.NET introduces a remarkable new model for creating web pages. In old-style web development, programmers had to master the quirks and details of HTML before they could design a dynamic web page. Pages had to be carefully tailored to a specific task, and the only way to generate additional content was to generate raw HTML tags.

ASP.NET solves this problem with a higher-level model of server controls. These controls are created and configured as objects. They run on the web server and they automatically provide their own HTML output. Even better, server controls behave like their Windows counterparts by maintaining state and raising events that you can react to in code.

In the previous unit , you built an exceedingly simple web page that incorporated a few controls you dragged in from the Visual Studio Toolbox. But before you create a more complex page, it's worth taking a step back to look at the big picture. ASP.NET actually provides two sets of server-side controls that you can incorporate into your web forms. These two different types of controls play subtly different roles:

1. ***HTML server controls:*** These are server-based equivalents for standard HTML elements. These controls are ideal if you're a seasoned web programmer who prefers to work with familiar HTML tags (at least at first). They are also useful when migrating ordinary HTML pages or ASP pages to ASP.NET, because they require the fewest changes.
2. ***Web controls:*** These are similar to the HTML server controls, but they provide a richer object model with a variety of properties for style and formatting details. They also provide more events and more closely resemble the controls used for Windows development. Web controls also feature some user interface elements that have no direct HTML equivalent, such as the GridView, Calendar, and validation controls.

Notes Even if you plan to use web controls exclusively, it's worth reading through this section to master the basics of HTML controls. Along the way, you'll get an introduction to a few ASP.NET essentials that apply to all kinds of server controls, including view state, postbacks, and event handling.

HTML Server Controls

HTML server controls provide an object interface for standard HTML elements. They provide three key features:

1. ***They generate their own interface***: You set properties in code, and the underlying HTML tag is created automatically when the page is rendered and sent to the client.
2. ***They retain their state***: Because the Web is stateless, ordinary web pages need to do a lot of work to store information between requests. HTML server controls handle this task automatically. For example, if the user selects an item in a list box, that item remains selected the next time the page is generated. Or, if your code changes the text in a button, the new text sticks the next time the page is posted back to the web server.
3. ***They fire server-side events***:

Example: buttons fire an event when clicked, text boxes fire an event when the text they contain is modified, and so on. Your code can respond to these events, just like ordinary controls in a Windows application. In ASP code, everything is grouped into one block that executes from

Notes

start to finish. With event-based programming, you can easily respond to individual user actions and create more structured code. If a given event doesn't occur, the event-handler code won't be executed.

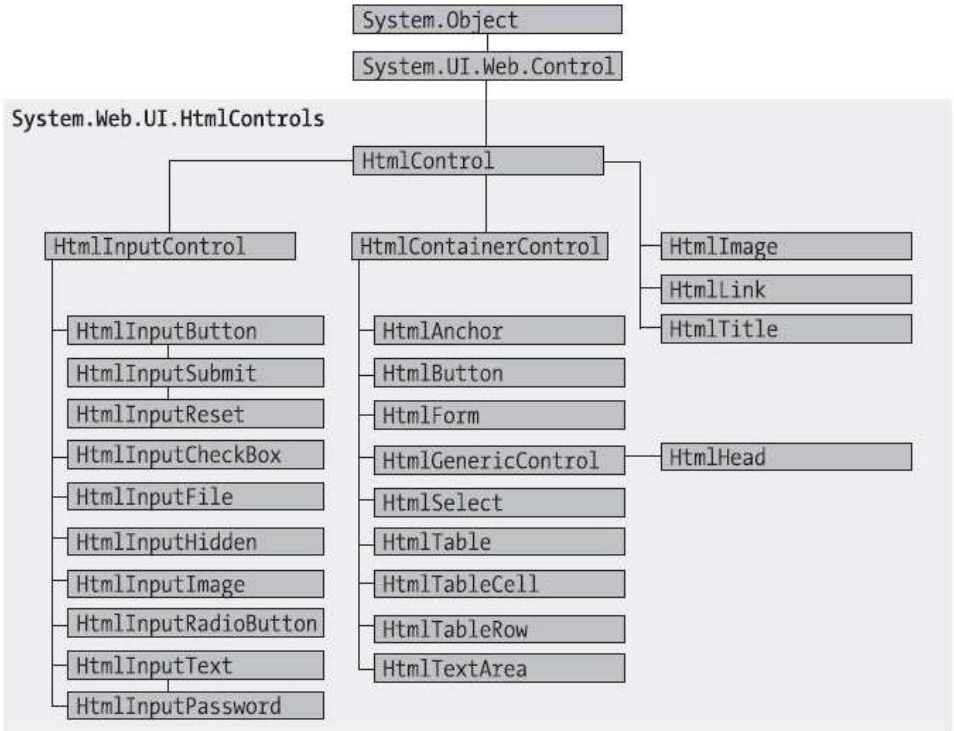
HTML server controls are ideal when you're performing a quick translation to add server-side code to an existing HTML page.

3.3 The Html Control Classes

Related classes in the .NET Framework use inheritance to share functionality.

Example: Every HTML control inherits from the base class HtmlControl. The HtmlControl class provides essential features every HTML server control uses. Figure 3.2 shows the inheritance diagram.

Figure 3.2: HTML Control Inheritance



Example: The HtmlImage class provides Align, Alt, Border, Src, Height, and Width properties. For this reason, users who are familiar with HTML syntax will find that HTML server controls are the most natural fit. Users who aren't as used to HTML will probably find that web controls have a more intuitive set of properties.

3.3.1 HTML Control Events

HTML server controls also provide one of two possible events: ServerClick or ServerChange.

The ServerClick is simply a click that's processed on the server side. It's provided by most button controls, and it allows your code to take immediate action. For example, consider the

Notes

HtmlAnchor control, which is the server control that represents the common HTML hyperlink (the <a> element). There are two ways to use the HtmlAnchor control. One option is to set its HtmlAnchor.HRef property to a URL, in which case the hyperlink will behave exactly like the ordinary HTML <a> element (the only difference being that you can set the URL dynamically in your code). The other option is to handle the HtmlAnchor.ServerClick event. In this case, when the link is clicked it will actually post back the page, allowing your code to run. The user won't be redirected to a new page unless you provide extra code to forward the request.

The ServerChange event responds when a change has been made to a text or selection control. This event isn't as useful as it appears because it doesn't occur until the page is posted back (for example, after the user clicks a submit button). At this point, the ServerChange event occurs for all changed controls, followed by the appropriate ServerClick. The Page.Load event is the first to fire, but you have no way to know the order of events for other controls.

Table 3.3 shows which controls provide a ServerClick event and which ones provide a ServerChange event.

Table 3.3: HTML Control Events

Event	Controls that provide it
ServerClick	HtmlAnchor, HtmlButton, HtmlInputButton, HtmlInputImage, HtmlInputReset
ServerChange	HtmlInputText, HtmlInputCheckBox, HtmlInputRadioButton, HtmlInputHidden, HtmlSelect, HtmlTextArea

3.3.2 Advanced Events with the HtmlInputImage Control

The first parameter identifies the object (in this case, the control) that fired the event. The second parameter is a special object that can include additional information about the event.

In the examples, the second parameter (e) has always been used to pass an empty System.EventArgs object. This object doesn't contain any additional information it's just a glorified placeholder. Here's one such example:

```
protected void Convert_ServerClick(Object sender, EventArgs e)
{ ... }
```

In fact, only one HTML server control sends additional information: the HtmlInputImage control. It sends an ImageClickEventArgs object (from the System.Web.UI namespace) that provides X and Y properties representing the location where the image was clicked. You'll notice that the definition for the HtmlInputImage.ServerClick event handler is a little different from the event handlers used with other controls:

```
protected void ImgButton_ServerClick(Object sender, ImageClickEventArgs e)
{ ... }
```

Using this additional information, you can replace multiple button controls and image maps with a single, intelligent HtmlInputImage control.

Here's the markup you need to create the HtmlInputImage control for this example:

```
<input type="image" ID="ImgButton" runat="server" src="button.png" />
OnServerClick="ImgButton_ServerClick" />
```

The sample ImageTest.aspx page shown in figure 3.3 puts this feature to work with a simple graphical button. Depending on whether the user clicks the button border or the button surface, a different message is displayed.

Notes

Figure 3.3: Using an HtmlInputImage Control

The page code examines the click coordinates provided by the ImageClickEventArgs object and displays them in another control. Here's the page code you need:

```
public partial class ImageTest : System.Web.UI.Page
{
protected void ImgButton_ServerClick(Object sender,
ImageClickEventArgs e)
{
Result.InnerText = "You clicked at (" + e.X.ToString() +
", " + e.Y.ToString() + "). ";
if ((e.Y < 100) && (e.Y > 20) && (e.X > 20) && (e.X < 275))
{
Result.InnerText += "You clicked on the button surface.";
}
else
{
Result.InnerText += "You clicked the button border.";
}
}
}
```

3.3.3 The HtmlControl Base Class

Every HTML control inherits from the base class HtmlControl. This relationship means that every HTML control will support a basic set of properties and features. Table 3.4 shows these properties.

Notes

Table 3.4: HtmlControl Properties

Property	Description
Attributes	Provides a collection of all the attributes that are set in the control tag, and their values. Rather than reading or setting an attribute through the Attributes, it's better to use the corresponding property in the control class. However, the Attributes collection is useful if you need to add or configure a custom attribute or an attribute that doesn't have a corresponding property.
Controls	Provides a collection of all the controls contained inside the current control. (For example, a server control could contain an <input> server control.) Each object is provided as a generic System.Web.UI.Control object so that you may need to cast the reference to access control-specific properties.
Disabled	Disables the control when set to true, thereby ensuring that the user cannot interact with it, and its events will not be fired.
EnableViewState	Disables the automatic state management for this control when set to false. In this case, the control will be reset to the properties and formatting specified in the control tag every time the page is posted back. If this is set to true (the default), the control uses a hidden input field to store information about its properties, thereby ensuring that any changes you make in code are remembered.
Page	Provides a reference to the web page that contains this control as a System.Web.UI.Page object.
Parent	Provides a reference to the control that contains this control. If the control is placed directly on the page (rather than inside another control), it will return a reference to the page object.
Style	Provides a collection of CSS style properties that can be used to format the control.
TagName	Indicates the name of the underlying HTML element (for example, img or div).
Visible	Hides the control when set to false and will not be rendered to the final HTML page that is sent to the client.

The HtmlControl class also provides built-in support for data binding.

Task

Write a HTML document that has a five-row table with the text in each column appearing as a different color.

3.3.4 The HtmlContainerControl Class

Any HTML control that requires a closing tag inherits from the HtmlContainer class (which in turn inherits from the more basic HtmlControl class). For example, elements such as <a>, <form>, and always use a closing tag, because they can contain other HTML elements. On the other hand, elements such as <img> and <input> are used only as stand-alone tags. Thus, the HtmlAnchor, HtmlForm, and HtmlGenericControl classes inherit from HtmlContainerControl, while HtmlInputImage and HtmlInputButton do not.

The HtmlContainer control adds two properties to those defined in HtmlControl, as described in Table 3.5.

Notes

Table 3.5: HtmlContainerControl Properties

Property	Description
InnerHtml	The HTML content between the opening and closing tags of the control. Special characters that are set through this property will not be converted to the equivalent HTML entities. This means you can use this property to apply formatting with nested tags such as <b>, <i>, and <h1>.
InnerText	The text content between the opening and closing tags of the control. Special characters will be automatically converted to HTML entities and displayed like text (for example, the less-than character (<) will be converted to < and will be displayed as < in the web page). This means you can't use HTML tags to apply additional formatting with this property. The simple currency converter page uses the InnerText property to enter results into a tag.

3.3.5 The HtmlInputControl Class

This control defines some properties (Table 3.6) that are used for the <input> element. As you've already learned, the <input> element can represent different controls, depending on the type attribute. The <input type="text"> element is a text box and <input type="submit"> is a button.

Table 3.6: HtmlInputControl Properties

Property	Description
Type	Provides the type of input control. For example, a control based on <input type="file"> would return file for the type property.
Value	Returns the contents of the control as a string. In the simple currency converter, this property allowed the code to retrieve the information entered in the text input control.

3.4 The Page Class

One control we haven't discussed in detail yet is the Page class. As explained in the previous unit, every web page is a custom class that inherits from System.Web.UI.Page. By inheriting from this class, your web page class acquires a number of properties and methods that your code can use.

Table 3.7 provides an overview of some of the more fundamental properties.

Table 3.7: Basic Page Properties

Property	Description
IsPostBack	This Boolean property indicates whether this is the first time the page is being run (false) or whether the page is being resubmitted in response to a control event, typically with stored view state information (true). You'll usually check this property in the Page.Load event handler to ensure that your initial web page initialization is only performed once.
EnableViewState	When set to false, this overrides the EnableViewState property of the contained controls, thereby ensuring that no controls will maintain state information.
Application	This collection holds information that's shared between all users in your website. For example, you can use the Application collection to count the number of times a page has been visited.
Session	This collection holds information for a single user, so it can be used in different pages. For example, you can use the Session collection to store the items in the current user's shopping basket on an e-commerce website.

Contd...

Notes

Cache	This collection allows you to store objects that are time-consuming to create so they can be reused in other pages or for other clients. This technique, when implemented properly, can improve performance of your web pages.
Request	This refers to an HttpRequest object that contains information about the current web request. You can use the HttpRequest object to get information about the user's browser, although you'll probably prefer to leave these details to ASP.NET.
Response	This refers to an HttpResponse object that represents the response ASP.NET will send to the user's browser.
Server	This refers to an HttpServerUtility object that allows you to perform a few miscellaneous tasks. For example, it allows you to encode text so that it's safe to place it in a URL or in the HTML markup of your page.
User	If the user has been authenticated, this property will be initialized with user information.

HTML Encoding

As you already know, there are certain characters that have a special meaning in HTML. For example, the angle brackets (< >) are always used to create tags. This can cause problems if you actually want to use these characters as part of the content of your web page.

Example: Imagine you want to display this text on a web page:

```
Enter a word <here>
```

If you try to write this information to a page or place it inside a control, you end up with this instead:

```
Enter a word
```

The problem is that the browser has tried to interpret the <here> as an HTML tag. A similar problem occurs if you actually use valid HTML tags. For example, consider this text:

```
To bold text use the <b> tag.
```

Not only will the text <b> not appear, but the browser will interpret it as an instruction to make the text that follows bold. To circumvent this automatic behavior, you need to convert potential problematic values to their HTML equivalents. For example, < becomes < in your final HTML page, which the browser displays as the < character. Table 3.8 lists some special characters that need to be encoded.

Table 3.8: Common HTML Special Characters

Result	Description	Encoded Entity
	Nonbreaking space	
<	Less-than symbol	<
>	Greater-than symbol	>
&	Ampersand	&
"	Quotation mark	"

You can performthis transformation on your own, or you can circumvent the problem by using the InnerText property of an HTML server control. When you set the contents of a control using InnerText, any illegal characters are automatically converted into their HTML equivalents. However, this won't help if you want to set a tag that contains a mix of embedded HTML tags and encoded characters. It also won't be of any use for controls that don't provide an InnerText property, such as the Label web control you'll examine in the next unit . In these cases, you can

Notes

use the HttpServerUtility.HtmlEncode() method to replace the special characters. (Remember, an HttpServerUtility object is provided through the Page.Server property.)

Example:

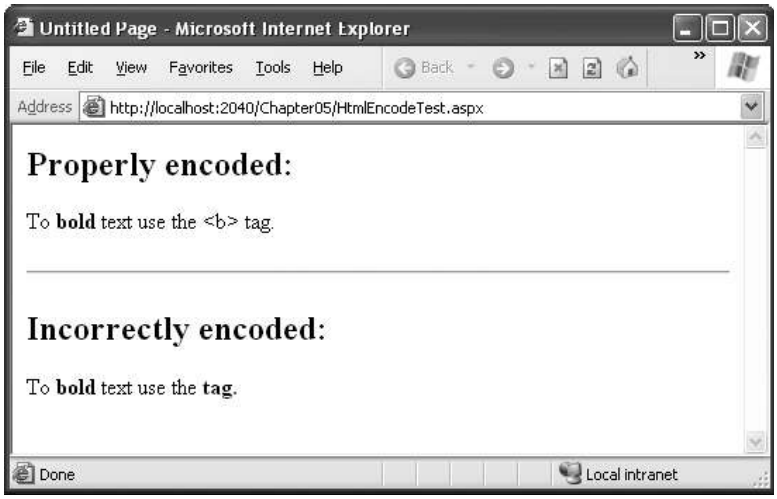
```
// Will output as "Enter a word &lt;here&gt;" in the HTML file, but the
// browser will display it as "Enter a word <here>".
ctrl.InnerHtml = Server.HtmlEncode("Enter a word <here>");
```

Or consider this example, which mingles real HTML tags with text that needs to be encoded:

```
ctrl.InnerHtml = "To <b>bold</b> text use the ";
ctrl.InnerHtml += Server.HtmlEncode("<b>") + " tag.";
```

Figure 3.4 shows the results of successfully and incorrectly encoding special HTML characters.

Figure 3.4: Encoding Special HTML Characters



The HtmlEncode() method is particularly useful if you're retrieving values from a database and you aren't sure whether the text is valid HTML. You can use the HtmlDecode() method to revert the text to its normal form if you need to perform additional operations or comparisons with it in your code.

Along with the HtmlEncode() and HtmlDecode() methods, the HttpServerUtility class also includes UrlEncode() and UrlDecode() methods. Much as HtmlEncode() allows you to convert text to valid HTML with no special characters, UrlEncode() allows you to convert text into a form that can be used in a URL. This technique is particularly useful if you want to pass information from one page to another by tacking it onto the end of the URL.

3.5 Application Events

Although server controls are the most common source of events, there's another type of event that you'll occasionally encounter: application events. Application events aren't nearly as important in an ASP.NET application as the events fired by server controls, but you might use them to perform additional processing tasks. For example, using application events you can write logging code that runs every time a request is received, no matter what page is being requested. Basic ASP.NET features like session state and authentication use application events to plug into the ASP.NET processing pipeline.

Notes

You can't handle application events in the code behind for a web form. Instead, you need the help of another ingredient: the Global.asax file.

3.5.1 The Global.asax File

The Global.asax file allows you to write code that responds to global application events. These events fire at various points during the lifetime of a web application, including when the application domain is first created (when the first request is received for a page in your website folder).

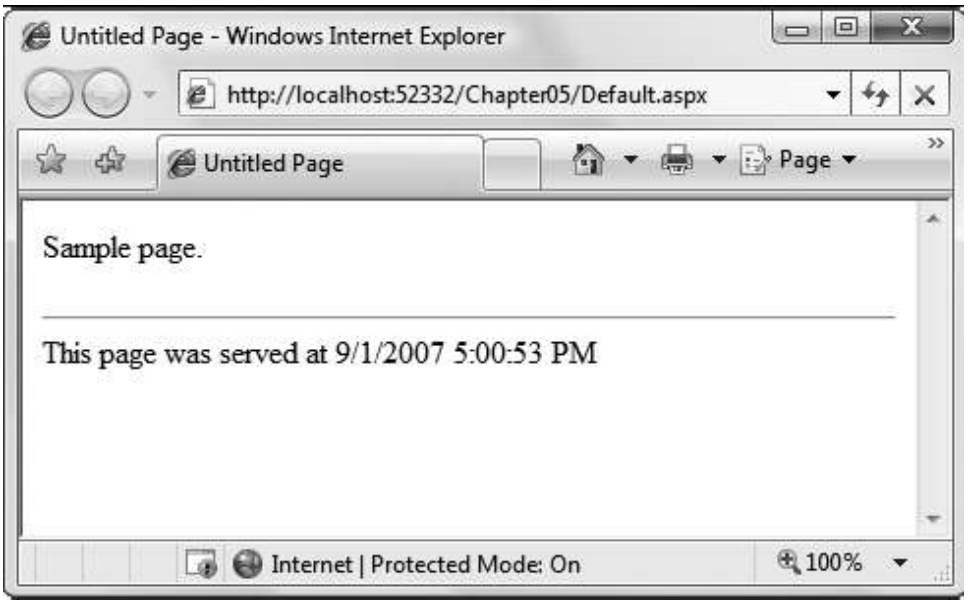
To add a Global.asax file to an application in Visual Studio, choose Website → Add New Item, and select the Global Application Class file type. Then, click OK.

The Global.asax file looks similar to a normal .aspx file, except that it can't contain any HTML or ASP.NET tags. Instead, it contains event handlers. For example, the following Global.asax file reacts to the Application.EndRequest event, which happens just before the page is sent to the user:

```
<%@ Application Language="C#" %>
<script language="c#" runat="server">
protected void Application_OnEndRequest()
{
Response.Write("<hr />This page was served at " +
DateTime.Now.ToString());
}
</script>
```

This event handler uses the Write() method of the built-in Response object to write a footer at the bottom of the page with the date and time that the page was created (Figure 3.5).

Figure 3.5: HelloWorld.aspx with an Automatic Footer



Notes

Each ASP.NET application can have one Global.asax file. Once you place it in the appropriate website directory, ASP.NET recognizes it and uses it automatically. For example, if you add the Global.asax file shown previously to a web application, every web page in that application will include a footer.

3.5.2 Additional Application Events

Application.EndRequest is only one of more than a dozen events you can respond to in your code. To create a different event handler, you simply need to create a subroutine with the defined name. Table 3.9 lists some of the most common application events that you'll use.

Table 3.9: Basic Application Events

Event-Handling Method	Description
Application_Start()	Occurs when the application starts, which is the first time it receives a request from any user. It doesn't occur on subsequent requests. This event is commonly used to create or cache some initial information that will be reused later.
Application_End()	Occurs when the application is shutting down, generally because the web server is being restarted. You can insert cleanup code here.
Application_BeginRequest()	Occurs with each request the application receives, just before the page code is executed.
Application_EndRequest()	Occurs with each request the application receives, just after the page code is executed.
Session_Start()	Occurs whenever a new user request is received and a session is started.
Session_End()	Occurs when a session times out or is programmatically ended. This event is only raised if you are using in-process session state storage (the InProc mode, not the StateServer or SQLServer modes).
Application_Error()	Occurs in response to an unhandled error.

3.6 ASP.NET Configuration

Every web application includes a web.config file that configures fundamental settings everything from the way error messages are shown to the security settings that lock out unwanted visitors.

The ASP.NET configuration files have several key advantages:

1. ***They are never locked:*** You can update web.config settings at any point, even while your application is running. If there are any requests currently under way, they'll continue to use the old settings, while new requests will get the changed settings right away.
2. ***They are easily accessed and replicated:*** Provided you have the appropriate network rights, you can change a web.config file from a remote computer. You can also copy the web.config file and use it to apply identical settings to another application or another web server that runs the same application in a web farm scenario.
3. ***The settings are easy to edit and understand:*** The settings in the web.config file are humanreadable, which means they can be edited and understood without needing a special configuration tool.

Notes

3.6.1 The web.config File

The web.config file uses a predefined XML format. The entire content of the file is nested in a root <configuration> element. Inside this element are several more subsections, some of which you'll never change, and others which are more important.

Here's the basic skeletal structure of the web.config file, with the three most important sections highlighted in bold:

```
<?xml version="1.0" ?>
<configuration>
<configSections>...</configSections>
<appSettings>...</appSettings>
<connectionStrings>...</connectionStrings>
<system.web>...</system.web>
<system.codedom>...</system.codedom>
<system.webServer>...</system.webServer>
</configuration>
```

Note that the web.config file is case-sensitive, like all XML documents, and starts every setting with a lowercase letter. This means you cannot write <AppSettings> instead of <appSettings>.

As a web developer, there are three sections in the web.config file that you'll work with. The <appSettings> section allows you to add your own miscellaneous pieces of information. The <connectionStrings> section allows you to define the connection information for accessing a database.

Inside the <system.web> element are separate elements for each aspect of website configuration. You can include as few or as many of these as you want. For example, if you need to specify special error settings, you would add the <customErrors> element in the <system.web> section. If you wanted to control how ASP.NET's security works, you'd add the <authentication> and <authorization> sections.

3.6.2 Nested Configuration

ASP.NET uses a multilayered configuration system that allows you to set settings at different levels.

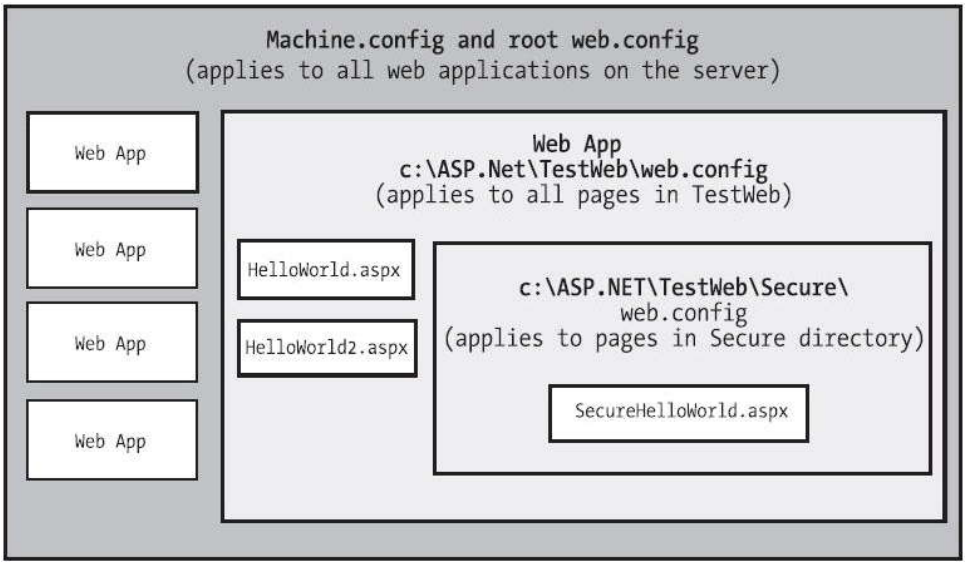
Every web server starts with some basic settings that are defined in two configuration files in the c:\Windows\Microsoft.NET\Framework\v2.0.50727\Config directory. These two files are machine.config and web.config. Generally, you won't edit either of these files manually, because they affect the entire computer. Instead, you'll configure the web.config file in your web application folder. Using that file, you can set additional settings or override the defaults that are configured in the two system files.

More interestingly, you can use different settings for different parts of your application. To use this technique, you need to create additional subdirectories inside your virtual directory. These subdirectories can contain their own web.config files with additional settings.

Subdirectories inherit web.config settings from the parent directory. For example, imagine you create a website in the directory c:\ASP.NET\TestWeb. Inside this directory, you create a folder named Secure. Pages in the c:\ASP.NET\TestWeb\Secure directory can acquire settings from three files, as shown in Figure 3.6.

Notes

Figure 3.6: Configuration Inheritance



Any machine.config or web.config settings that aren't explicitly overridden in the c:\ASP.NET\TestWeb\Secure\web.config file will still apply to the SecureHelloWorld.aspx page. In this way, subdirectories can specify just a small set of settings that differ from the rest of the web application. One reason you might want to use multiple directories in an application is to apply different security settings. Files that need to be secured would then be placed in a dedicated directory with a web.config file that defines more stringent security settings.

3.6.3 Storing Custom Settings in the web.config File

ASP.NET also allows you to store your own settings in the web.config file, in an element called <appSettings>. Note that the <appSettings> element is nested in the root <configuration> element. Here's the basic structure:

```
<?xml version="1.0" ?>
<configuration>
...
<appSettings>
<!-- Custom application settings go here. -->
</appSettings>
...
<system.web>
<!-- ASP.NET Configuration sections go here. -->
</system.web>
...
</configuration>
```

Notes This example adds a comment in the place where you'd normally find additional settings. XML comments are bracketed with the <!-- and --> character sequences. You can also use XML comments to temporarily disable a setting in a configuration file.

Notes

The custom settings that you add are written as simple string variables. You might want to use a special web.config setting for several reasons:

1. ***To centralize an important setting that needs to be used in many different pages:*** For example, you could create a variable that stores a database query. Any page that needs to use this query can then retrieve this value and use it.
2. ***To make it easy to quickly switch between different modes of operation:*** For example, you might create a special debugging variable. Your web pages could check for this variable and, if it's set to a specified value, output additional information to help you test the application.
3. ***To set some initial values:*** Depending on the operation, the user might be able to modify these values, but the web.config file could supply the defaults.

You can enter custom settings using an <add> element that identifies a unique variable name (key) and the variable contents (value). The following example adds a variable that defines a file path where important information is stored:

```
<appSettings>
<add key="DataFilePath"
value="e:\NetworkShare\Documents\WebApp\Shared" />
</appSettings>
```

You can add as many application settings as you want, although this example defines just one.

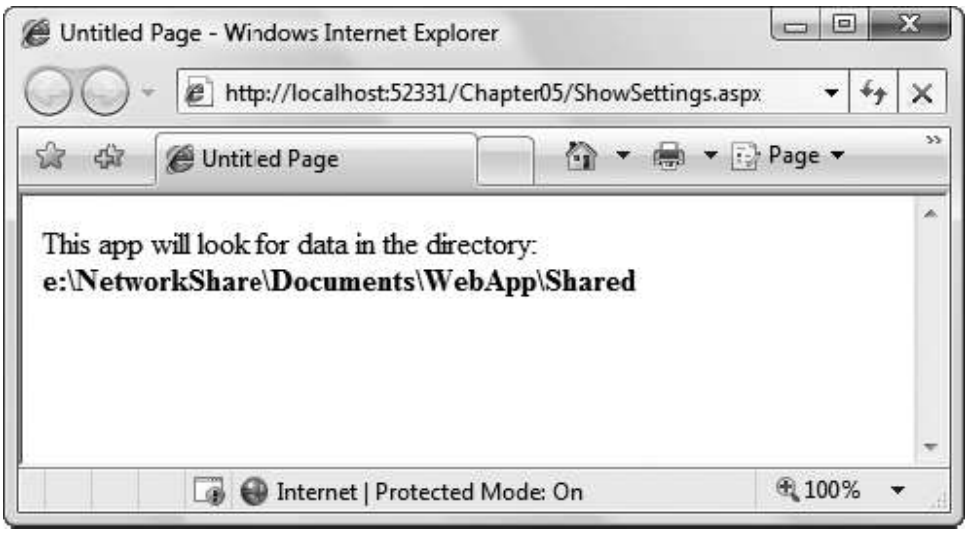
You can create a simple test page to query this information and display the results, as shown in the following example (which is provided with the sample code as ShowSettings.aspx and ShowSettings.aspx.cs). You retrieve custom application settings from web.config by key name, using the WebConfigurationManager class, which is found in the System.Web.Configuration namespace. This class provides a static property called AppSettings with a collection of application settings.

```
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.Configuration;
public partial class ShowSettings : System.Web.UI.Page
{
protected void Page_Load()
{
lblTest.Text = "This app will look for data in the directory:<br /><b>";
lblTest.Text += WebConfigurationManager.AppSettings["DataFilePath"];
lblTest.Text += "</b>";
}
}
```

The simple application just displays the custom web.config setting, as shown in Figure 3.7.

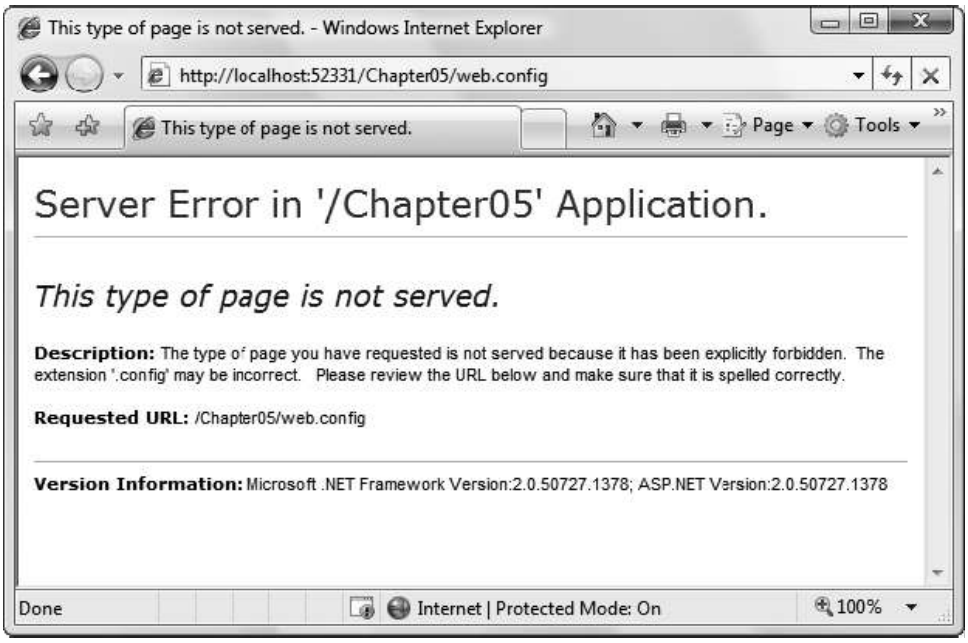
Notes

Figure 3.7: Displaying Custom Application Settings



ASP.NET is configured, by default, to deny any requests for .config files. This means a remote user will not be able to access the file through IIS. Instead, they'll receive the error message shown in Figure 3.8.

Figure 3.8: Requests for web.config are denied



Task

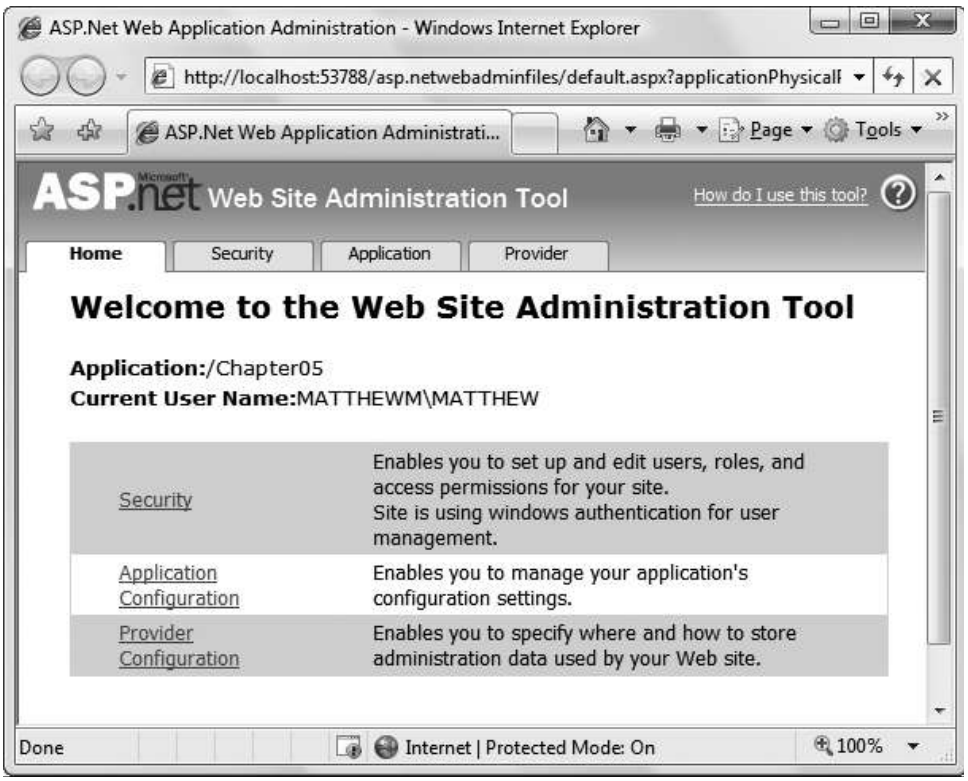
Build a text paragraph that uses a fixed-width font versus the regular browser font. Try adding in some text effects to see how the text is displayed differently.

Notes

3.6.4 The Website Administration Tool (WAT)

Editing the web.config file by hand is refreshingly straightforward, but it can be a bit tedious. To help alleviate the drudgery, ASP.NET includes a graphical configuration tool called the Website Administration Tool (WAT), which lets you configure various parts of the web.config file using a web page interface. To run the WAT to configure the current web project in Visual Studio, select Website → ASP.NET Configuration. A web browser window will appear (Figure 3.9). Internet Explorer will automatically log you on under the current Windows user account, allowing you to make changes.

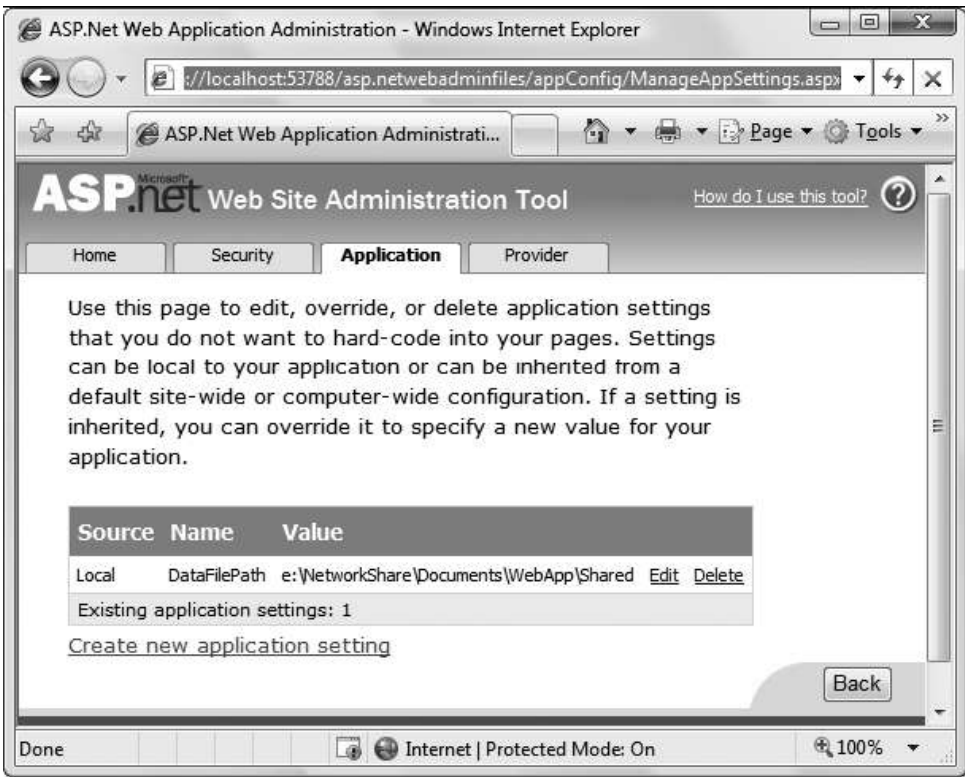
Figure 3.9: Running the WAT



You can use the WAT to automate the web.config changes you made in the previous example. To try this, click the Application tab. Using this tab, you can create a new setting (click the Create Application Settings link). If you click Manage Application Settings, you'll see a list with all the applications settings that are defined in your application Figure 3.10. You can then choose to remove or edit any one of them.

Notes

Figure 3.10: Editing an Application Setting with the WAT



This is the essential idea behind the WAT. You make your changes using a graphical interface (a web page), and the WAT generates the settings you need and adds them to the web.config file for your application behind the scenes. Of course, the WAT has a number of settings for configuring more complex ASP.NET settings.

Case Study

Online Survey Company Embraces Growth with Flexible, Scalable IT Technology

SurveyMonkey empowers people to conduct self-service market research with tools that enable the quick and easy creation of online surveys. Through its Web site at SurveyMonkey.com, the company provides an easy-to-use interface that lets users rapidly create custom surveys and then distribute them over the Internet to their selected audience. The results, which are stored by SurveyMonkey, can be viewed or printed out in reports.

Since entering the online survey market in 1999, SurveyMonkey has experienced tremendous growth. The company consistently ranks among the top 800 most visited sites on the Internet, as ranked by Alexa, the Web traffic tool. This growth is due not only to the ease of use provided by the service, but also to the low cost—SurveyMonkey provides a full-featured "professional subscription" for only U.S.$20 a month. Basic subscriptions are free.

Contd...

Notes

From the beginning, SurveyMonkey.com was built using various Microsoft® technologies, including the Windows Server® operating system, Active Server Pages (ASP) technology, Microsoft SQL Server™ 2000, and tools in the Visual Studio® development system. Part of the appeal of using Microsoft tools was to provide a familiar development environment.

Over the course of about six years, the company grew from an unknown startup to a leader in the online survey market. SurveyMonkey now gets about 60 million page views per month and daily activity that includes more than 2,500 surveys created and 100,000 survey responses.

By early 2005, SurveyMonkey began evaluating the company's IT infrastructure to make sure that its technology could keep up with its rapid growth.

Solution:

SurveyMonkey decided to upgrade and enhance its IT system by employing Microsoft .NET software for connecting people, information, systems, and devices. As part of the upgrade, the company migrated its database to SQL Server 2005 to take advantage of the 64-bit computing capabilities of the latest version; Microsoft ASP.NET version 2.0 for creating and updating the SurveyMonkey Web site; and the Visual C#® 2005 development tool. The development tools are part of the Microsoft .NET Framework, an integral component of the Windows® operating system that provides a programming model and runtime for Web services, Web applications, and smart client applications.

The enhanced IT system runs on Windows Server 2003 Web Edition. A Citrix NetScaler application delivery solution running in an active/passive pair helps to balance server loads during peak periods.

Benefits

By using the Microsoft .NET software, SurveyMonkey can scale its Web site to meet the growing daily demand for its service while continuing to provide value to its customers. The Microsoft development tools help SurveyMonkey continually adjust and improve its service without adding people to its staff. Microsoft technology also provides the flexibility that enables SurveyMonkey to quickly and easily add the capabilities of third-party software products without having to invest in major development efforts.

Greater Scalability

Ryan Finley, President of SurveyMonkey, says the company has succeeded in increasing its customer base primarily through "viral marketing," which relies largely on word-of-mouth recommendations instead of big advertising and marketing budgets. The Microsoft software that SurveyMonkey implemented helps the company keep up with the exponentially increasing demand for its services.

"We see a lot of value in SQL Server 2005, especially its ability to take advantage of 64 bit processing power," Finley says. SQL Server 2005 uses advanced memory-addressing capabilities for computing resources such as buffer pools and caches, which reduces the need to move data in and out of the hard disk into RAM.

Finley explains, "With 32-bit software, we were limited to using 4 gigabytes of memory in our server hardware, but that all changes with 64-bit computing. We currently have about 800 gigabytes of data from surveys stored on the servers. That will continue to grow, and the ability of SQL Server 2005 to take advantage of 64-bit processing will help improve the overall performance that is experienced by our users."

Contd...

Notes

Scalability is important for another reason. The company's customers are increasing not only the size of individual surveys, but also the sheer number of surveys conducted.

Fred Meyers, President of The Queensboro Shirt Company in Wilmington, North Carolina, says the ability to conduct many surveys at a low price is what keeps him coming back to SurveyMonkey.

"We've done about 50,000 customer surveys through SurveyMonkey. We custom embroider logos on a wide variety of apparel and accessories, and we introduce a lot of new products on a regular basis. We are able to plan our products and make a lot of tweaks based on customer feedback in the surveys," Meyers says. "SurveyMonkey is a really valuable tool. I spend just a small amount of money each month and get an enormous amount of feedback. It's one of the great values in all of business today."

Easier Development

The rich and flexible Microsoft development tools are helping the company continually enhance and modify the SurveyMonkey services without straining any of the company's development resources, says Finley.

"With the new programming environment, we are discovering that we can reuse a lot of code, which has really streamlined our development processes," he says.

"For example, we use classes that we've created in Visual C#, so when we want to add a new feature to the site, we don't have to worry about how it will interact with other elements. It just works," Finley says. "The Microsoft tools give us a full-fledged development environment in which a couple of people can work on the same project, and they don't have to worry about messing up someone else's code."

Greater Flexibility

Along with gaining a powerful development platform, Finley says, SurveyMonkey also benefits from the IT environment's compatibility with a wide range of third-party software applications, which can be easily added as necessary to improve the company's services.

"On occasion," he explains, "we have wanted to add new features that will enhance the site—for example, adding graphing capabilities so users can get a visual representation of the numbers gathered in their surveys. The great thing about the Microsoft development environment is that all the tools we need to easily add new software are already in place. We're going to be adding a charting application from Dundas and will be using ScaleOut Software, which integrates with ASP.NET and enables caching for faster throughput."

The power and flexibility of the Microsoft tools and technologies, Finley adds, are helping SurveyMonkey continue to attract new customers.

"A large part of our success is due to being able to strike a balance between a full-featured survey tool and an easy-to-use survey tool," he says. "A big advantage that we offer is a high level of usability that lets people who don't normally do market research to get started immediately. It is all point and click, easy to navigate.

"In this market, there are a lot of very high-end products that cost tens of thousands of dollars," Finley continues. "We think a big part of the market for survey tools is fed up with paying so much. For a vast majority of companies and individuals, what we provide is plenty for what they need. The Microsoft technology is helping us reach our customers with software that is powerful yet easy to use, and cost-effective enough for us to maintain a compelling, Web-based service."

Notes

3.7 Summary

- This unit presented you with your first look at web applications, web pages, and configuration.
- You should now understand how to create an ASP.NET web page and use HTML server controls.
- HTML controls are a compromise between web controls and traditional ASP.NET programming. They use the familiar HTML elements but provide a limited object-oriented interface.
- Essentially, HTML controls are designed to be straightforward, predictable, and automatically compatible with existing programs.
- With HTML controls, the final HTML page that is sent to the client closely resembles the original .aspx page.

3.8 Keywords

.NET Framework Data: A component of ADO.NET that provides access to data from a relational data source. A .NET Framework data provider contains classes to connect to a data source, execute commands at the data source, and return query results from the data source, including the ability to execute commands within transactions.

Active Server Pages (ASP): A Microsoft technology for creating server-side, Web-based application services. ASP applications are typically written using a scripting language, such as JScipt, VBScript, or PerlScript.

Application Domain: A virtual process space within which ASP.NET is hosted and executed. On a Web server, it is possible to have multiple ASP.NET Web applications running inside a single process.

ASP.NET State Server: A Windows service that provides a default server implementation of the ASP.NET State Server Protocol. When the service is enabled on a computer, that computer can act as a state server. The state server accepts requests to load, store, delete, and temporarily lock Session state items.

Web Application Identifier: Each ASP.NET application running on a Web server is uniquely identified with a Web application identifier.

3.9 Self Assessment

Fill in the blanks:

1. Every ..... application shares a common set of resources and configuration settings.
2. Every ..... should have a well-planned directory structure.
3. ..... provide one of two possible events: ServerClick or ServerChange.
4. Every HTML control inherits from the base class .....
5. The ..... method is particularly useful if you're retrieving values from a database and you aren't sure whether the text is valid HTML.
6. ASP.NET features like ..... and authentication use application events to plug into the ASP.NET processing pipeline.

Notes

State whether the following statements are True or False:

7. The Global.asax file allows you to write code that responds to global application events.
8. Application.EndRequest is not only one of more than a dozen events you can respond to in your code.
9. The web.config file uses a predefined XML format.
10. ASP.NET allows you to store your own settings in the web.config file, in an element called <appSettings>.

3.10 Review Questions

1. What is the basic pattern for showing data on an ASP.NET 3.5 page?
2. Name several types of data sources that ASP.NET 3.5 supports.
3. Name several ways that ASP.NET 3.5 can display data.
4. What are the differences between the .NET Framework 3.5 and ASP.NET 2.0?
5. Make some observations comparing SQL Server, MSDE, and SSE.
6. Describe various ASP.NET files in detail.
7. What do you mean by ASP.NET directories?
8. Explain HTML server control.
9. What do you mean by page class?
10. Write short note on HTML encoding.

Answers: Self Assessment

1.	ASP.NET	2.	web application
3.	HTML server controls	4.	HtmlControl
5.	HtmlEncode()	6.	session state
7.	True	8.	False
9.	True	10.	True

3.11 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Notes

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Notes

Unit 4: Web Controls

CONTENTS

Objectives

Introduction

4.1 Web Controls

4.2 Basic Web Control Classes

4.3 Web Control Classes

4.3.1 The WebControl Base Class

4.3.2 The Default Button

4.4 List Controls

4.4.1 Multiple-Select List Controls

4.4.2 The BulletedList Control

4.4.3 Table Controls

4.5 Web Control Events and Autopostback

4.6 A Simple Web Page

4.7 Summary

4.8 Keywords

4.9 Self Assessment

4.10 Review Questions

4.11 Further Readings

Objectives

After studying this unit, you will be able to:

- Explain web control classes
- Define list controls
- Describe web control events
- Know a simple web page

Introduction

HTML server controls really show only a glimpse of what is possible with ASP.NET's server control model. To see some of the real advantages, you need to dive into the richer and more extensible web controls. In this unit, you'll explore the basic web controls and their class hierarchy. You'll also delve deeper into ASP.NET's event handling, learn the details of the web page life cycle, and put your knowledge to work by creating a web page that lets the user design a greeting card.

Notes

4.1 Web Controls

Now that you've seen the new model of server controls, you might wonder why you need additional web controls. But in fact, HTML controls are much more limited than server controls need to be.

Example: Every HTML control corresponds directly to an HTML tag, meaning you're bound by the limitations and abilities of HTML. Web controls, on the other hand, have no such restriction. They emphasize the future of web design.

These are some of the reasons you should switch to web controls:

1. ***They provide a rich user interface:*** A web control is programmed as an object but doesn't necessarily correspond to a single element in the final HTML page. For example, you might create a single Calendar or GridView control, which will be rendered as dozens of HTML elements in the final page. When using ASP.NET programs, you don't need to know anything about HTML. The control creates the required HTML tags for you.
2. ***They provide a consistent object model:*** HTML is full of quirks and idiosyncrasies. For example, a simple text box can appear as one of three elements, including <textarea>, <input type="text">, and <input type="password">. With web controls, these three elements are consolidated as a single TextBox control. Depending on the properties you set, the underlying HTML element that ASP.NET renders may differ. Similarly, the names of properties don't follow the HTML attribute names. For example, controls that display text, whether it's a caption or a text box that can be edited by the user, expose a Text property.
3. ***They tailor their output automatically:*** ASP.NET server controls can detect the type of browser and automatically adjust the HTML code they write to take advantage of features such as support for JavaScript. You don't need to know about the client because ASP.NET handles that layer and automatically uses the best possible set of features. This feature is known as adaptive rendering.
4. ***They provide high-level features:*** You'll see that web controls allow you to access additional events, properties, and methods that don't correspond directly to typical HTML controls. ASP.NET implements these features by using a combination of tricks.

4.2 Basic Web Control Classes

If you've ever created a Windows application before, you're probably familiar with the basic set of standard controls, including labels, buttons, and text boxes. ASP.NET provides web controls for all these standbys. (And if you've created .NET Windows applications, you'll notice that the class names and properties have many striking similarities, which are designed to make it easy to transfer the experience you acquire in one type of application to another.)

Table 4.1 lists the basic control classes and the HTML elements they generate. Some controls (such as Button and TextBox) can be rendered as different HTML elements. In this case, ASP.NET uses the element that matches the properties you've set. Also, some controls have no real HTML equivalent. For example, the CheckBoxList and RadioButtonList controls output as a <table> that contains multiple HTML check boxes or radio buttons. ASP.NET exposes them as a single object on the server side for convenient programming, thus illustrating one of the primary strengths of web controls.

Notes

Table 4.1: Basic Web Controls

Control Class	Underlying HTML Element
Label	
Button	<input type="submit"> or <input type="button">>
TextBox	<input type="text">, <input type="password">, or <textarea>
CheckBox	<input type="checkbox">>
RadioButton	<input type="radio">>
Hyperlink	<a>
LinkButton	<a> with a contained <img> tag
ImageButton	<input type="image">>
Image	<img>
ListBox	<select size="X"> where X is the number of rows that are visible at once
DropDownList	<select>
CheckBoxList	A list or <table> with multiple <input type="checkbox"> tags
RadioButtonList	A list or <table> with multiple <input type="radio"> tags
BulletedList	An ordered list (numbered) or unordered list (bulleted)
Panel	
Table, TableRow, and	<table>, <tr>, and <td> or <th>
TableCell	

This table omits some of the more specialized controls used for data, navigation, security, and web portals.

The Web Control Tags

ASP.NET tags have a special format. They always begin with the prefix asp: followed by the class name. If there is no closing tag, the tag must end with />. Each attribute in the tag corresponds to a control property, except for the runat="server" attribute, which signals that the control should be processed on the server.

The following, for example, is an ASP.NET TextBox:

```
<asp:TextBox ID="txt" runat="server" />
```

When a client requests this .aspx page, the following HTML is returned. The name is a special attribute that ASP.NET uses to track the control.

```
<input type="text" ID="txt" name="txt" />
```

Alternatively, you could place some text in the TextBox, set its size, make it read-only, and change the background color. All these actions have defined properties. For example, the TextBox.TextMode property allows you to specify SingleLine (the default), MultiLine (for a <textarea> type of control), or Password (for an input control that displays bullets to hide the true value). You can adjust the color using the BackColor and ForeColor properties. And you can tweak the size of the TextBox using the Rows property.

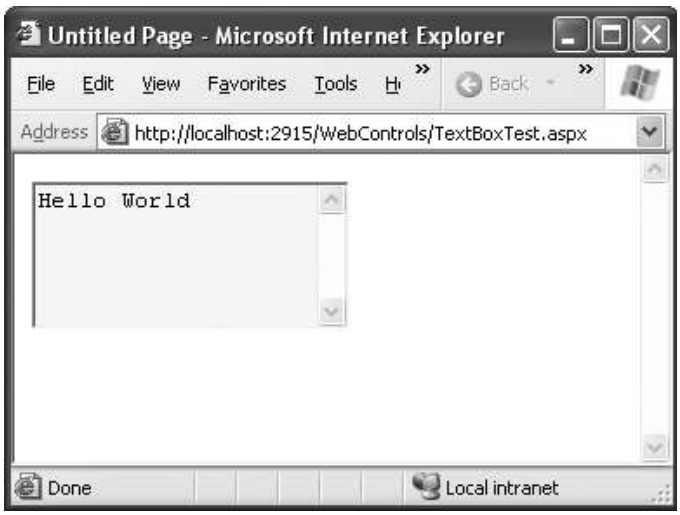
Example: **customized TextBox:**

```
<asp:TextBox ID="txt" BackColor="Yellow" Text="Hello World"
ReadOnly="True" TextMode="MultiLine" Rows="5" runat="server" />
```

The resulting HTML uses the <textarea> element and sets all the required style attributes. Figure 4.1 shows it in the browser.

Notes

Figure 4.1: A Customized Text Box



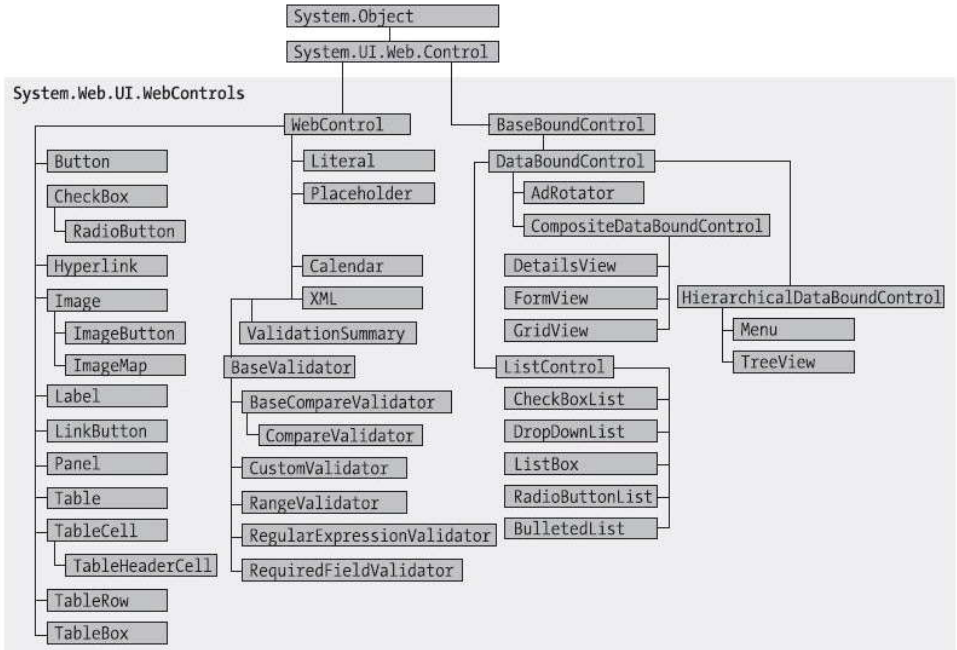

```
<textarea name="txt" rows="5" cols="20" readonly="readonly" ID="txt"
style="background-color:Yellow;">Hello World</textarea>
```

Clearly, it's easy to create a web control tag. It doesn't require any understanding of HTML. However, you will need to understand the control class and the properties that are available to you.

4.3 Web Control Classes

Web control classes are defined in the System.Web.UI.WebControls namespace. They follow a slightly more tangled object hierarchy than HTML server controls, as shown in Figure 4.2.

Figure 4.2: The Web Control Hierarchy



Notes

4.3.1 The WebControl Base Class

Most web controls begin by inheriting from the WebControl base class. This class defines the essential functionality for tasks such as data binding and includes some basic properties that you can use with almost any web control, as described in Table 4.2.

Table 4.2: WebControl Properties

Property	Description
AccessKey	Specifies the keyboard shortcut as one letter. For example, if you set this to Y, the Alt+Y keyboard combination will automatically change focus to this web control. This feature is supported only on Internet Explorer 4.0 and higher.
BackColor, ForeColor,and BorderColor	Sets the colors used for the background, foreground, and border of the control. In most controls, the foreground color sets the text color.
BorderWidth	Specifies the size of the control border.
BorderStyle	One of the values from the BorderStyle enumeration, including Dashed, Dotted, Double, Groove, Ridge, Inset, Outset, Solid, and None.
Controls	Provides a collection of all the controls contained inside the current control. Each object is provided as a generic System.Web.UI.Control object, so you will need to cast the reference to access control-specific properties.
Enabled	When set to false, the control will be visible, but it will not be able to receive user input or focus.
EnableViewState	Set this to false to disable the automatic state management for this control. In this case, the control will be reset to the properties and formatting specified in the control tag (in the .aspx page) every time the page is posted back. If this is set to true (the default), the control uses the hidden input field to store information about its properties, ensuring that any changes you make in code are remembered.
Font	Specifies the font used to render any text in the control as a System.Web.UI.WebControls.FontInfo object.
Height and Width	Specifies the width and height of the control. For some controls, these properties will be ignored when used with older browsers.
Page	Provides a reference to the web page that contains this control as a System.Web.UI.Page object.
Parent	Provides a reference to the control that contains this control. If the control is placed directly on the page (rather than inside another control), it will return a reference to the page object.
TabIndex	A number that allows you to control the tab order. The control with a TabIndex of 0 has the focus when the page first loads. Pressing Tab moves the user to the control with the next lowest TabIndex, provided it is enabled. This property is supported only in Internet Explorer 4.0 and higher.
ToolTip	Displays a text message when the user hovers the mouse above the control. Many older browsers don't support this property.
Visible	When set to false, the control will be hidden and will not be rendered to the final HTML page that is sent to the client.

Units

All the properties that use measurements, including BorderWidth, Height, and Width, require the Unit structure, which combines a numeric value with a type of measurement (pixels, percentage, and so on). This means when you set these properties in a control tag, you must make sure to append px (pixel) or % (for percentage) to the number to indicate the type of unit.

Here's an example with a Panel control that is 300 pixels wide and has a height equal to 50 percent of the current browser window:

```
<asp:Panel Height="300px" Width="50%" ID="pnl" runat="server" />
```

Notes

If you're assigning a unit-based property through code, you need to use one of the static methods of the Unit type. Use Pixel() to supply a value in pixels, and use Percentage() to supply a percentage value:

```
// Convert the number 300 to a Unit object
// representing pixels, and assign it.
pnl.Height = Unit.Pixel(300);
// Convert the number 50 to a Unit object
// representing percent, and assign it.
pnl.Width = Unit.Percentage(50);
```

You could also manually create a Unit object and initialize it using one of the supplied constructors and the UnitType enumeration. This requires a few more steps but allows you to easily assign the same unit to several controls:

```
// Create a Unit object.
Unit myUnit = new Unit(300, UnitType.Pixel);

// Assign the Unit object to several controls or properties.
pnl.Height = myUnit;
pnl.Width = myUnit;
```

Enumerations

Enumerations are used heavily in the .NET class library to group a set of related constants. For example, when you set a control's BorderStyle property, you can choose one of several predefined values from the BorderStyle enumeration. In code, you set an enumeration using the dot syntax:

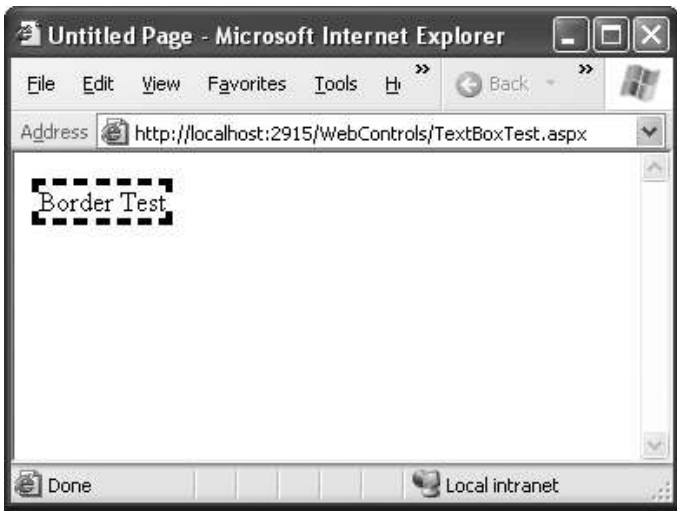
```
ctrl.BorderStyle = BorderStyle.Dashed;
```

In the .aspx file, you set an enumeration by specifying one of the allowed values as a string. You don't include the name of the enumeration type, which is assumed automatically.

```
<asp:Label BorderStyle="Dashed" Text="Border Test" ID="lbl"
runat="server" />
```

Figure 4.3 shows the label with the altered border.

Figure 4.3: Modifying the Border Style



Notes

Colors

The Color property refers to a Color object from the System.Drawing namespace. You can create color objects in several ways:

1. ***Using an ARGB (alpha, red, green, blue) color value:*** You specify each value as an integer from 0 to 255. The alpha component represents the transparency of a color, and usually you'll use 255 to make the color completely opaque.
2. ***Using a predefined .NET color name:*** You choose the correspondingly named read-only property from the Color structure. These properties include the 140 HTML color names.
3. ***Using an HTML color name:*** You specify this value as a string using the ColorTranslator class.

To use any of these techniques, you'll probably want to start by importing the System.Drawing namespace, as follows:

```
using System.Drawing;
```

The following code shows several ways to specify a color in code:

```
// Create a color from an ARGB value
int alpha = 255, red = 0, green = 255, blue = 0;
ctrl.ForeColor = Color.FromArgb(alpha, red, green, blue);

// Create a color using a .NET name
ctrl.ForeColor = Color.Crimson;

// Create a color from an HTML code
ctrl.ForeColor = ColorTranslator.FromHtml("Blue");
```

When defining a color in the .aspx file, you can use any one of the known color names:

```
<asp:TextBox ForeColor="Red" Text="Test" ID="txt" runat="server" />
```

The HTML color names that you can use are listed in the MSDN Help. Alternatively, you can use a hexadecimal color number (in the format #<red><green><blue>) as shown here:

```
<asp:TextBox ForeColor="#ff50ff" Text="Test"
ID="txt" runat="server" />
```

Fonts

The Font property actually references a full FontInfo object, which is defined in the System.Web.UI.WebControls namespace. Every FontInfo object has several properties that define its name, size, and style (Table 4.3).

Notes

Table 4.3: FontInfo Properties

Property	Description
Name	A string indicating the font name (such as Verdana).
Names	An array of strings with font names, in the order of preference. The browser will use the first matching font that's installed on the user's computer.
Size	An array of strings with font names, in the order of preference. The browser will use the first matching font that's installed on the user's computer.
Bold, Italic, Strikeout, Underline, and Overline	Boolean properties that apply the given style attribute.

In code, you can assign a font by setting the various font properties using the familiar dot syntax:

```
ctrl.Font.Name = "Verdana";
ctrl.Font.Bold = true;
```

You can also set the size using the FontUnit type:

```
// Specifies a relative size.
ctrl.Font.Size = FontUnit.Small;
// Specifies an absolute size of 14 pixels.
ctrl.Font.Size = FontUnit.Point(14);
```

In the .aspx file, you need to use a special "object walker" syntax to specify object properties such as Font. The object walker syntax uses a hyphen (-) to separate properties. For example, you could set a control with a specific font (Tahoma) and font size (40 point) like this:

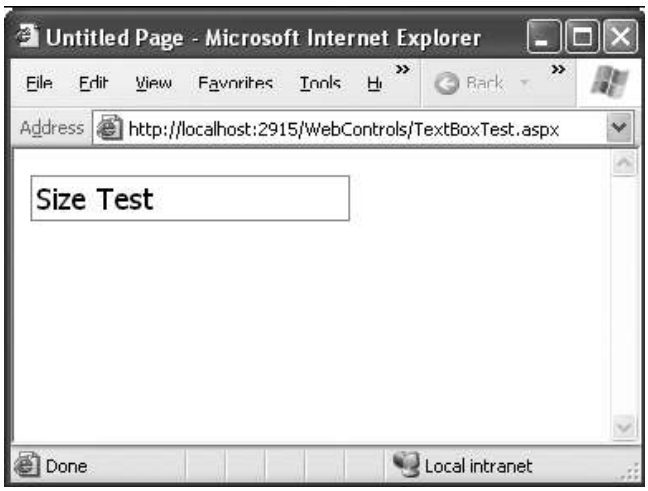
```
<asp:TextBox Font-Name="Tahoma" Font-Size="40" Text="Size Test" ID="txt"
runat="server" />
```

Or you could set a relative size like this:

```
<asp:TextBox Font-Name="Tahoma" Font-Size="Large" Text="Size Test"
ID="txt" runat="server" />
```

Figure 4.4 shows the altered TextBox in this example.

Figure 4.4: Modifying a Control's Font



Notes

A font setting is really just a recommendation. If the client computer doesn't have the font you request, it reverts to a standard font. To deal with this problem, it's common to specify a list of fonts, in order of preference. To do so, you use the Font.Names property instead of Font.Name, as shown here:

```
<asp:TextBox Font-Names="Verdana,Tahoma,Arial"
Text="Size Test" ID="txt" runat="server" />
```

Here, the browser will use the Verdana font (if it has it). If not, it will fall back on Tahoma or Arial.

When specifying fonts, it's a good idea to end with one of the following fonts, which are supported on all browsers:

1. Times
2. Arial and Helvetica
3. Courier

The following fonts are found on almost all Windows and Mac computers, but not necessarily on other operating systems like Unix:

1. Verdana
2. Georgia
3. Tahoma
4. Comic Sans
5. Arial Black
6. Impact

4.3.2 The Default Button

Along with control focusing, ASP.NET also allows you to designate a default button on a web page. The default button is the button that is "clicked" when the user presses the Enter key. For example, if your web page includes a form, you might want to make the submit button into a default button. That way, if the user hits Enter at any time, the page is posted back and the Button.Click event is fired for that button.

To designate a default button, you must set the HtmlForm.DefaultButton property with the ID of the respective control, as shown here:

```
<form DefaultButton="cmdSubmit" runat="server">
```

The default button must be a control that implements the IButtonControl interface. The interface is implemented by the Button, LinkButton, and ImageButton web controls but not by any of the HTML server controls.

In some cases, it makes sense to have more than one default button. For example, you might create a web page with two groups of input controls. Both groups may need a different default button. You can handle this by placing the groups into separate panels. The Panel control also exposes the DefaultButton property, which works when any input control it contains gets the focus.

Notes

Task Build a web page that will allow the user to sample some wave files based on a button click provided for each. Use an HTML table to provide a nice interface.

4.4 List Controls

The list controls include the ListBox, DropDownList, CheckBoxList, RadioButtonList, and BulletedList. They all work in essentially the same way but are rendered differently in the browser. The ListBox, for example, is a rectangular list that displays several entries, while the DropDownList shows only the selected item. The CheckBoxList and RadioButtonList are similar to the ListBox, but every item is rendered as a check box or option button, respectively. Finally, the BulletedList is the odd one out it's the only list control that isn't selectable. Instead, it renders itself as a sequence of numbered or bulleted items.

All the selectable list controls provide a SelectedIndex property that indicates the selected row as a zero-based index.

Example: If the first item in the list is selected, the SelectedIndex will be 0. Selectable list controls also provide an additional SelectedItem property, which allows your code to retrieve the ListItem object that represents the selected item. The ListItem object provides three important properties: Text (the displayed content), Value (the hidden value from the HTML markup), and Selected (true or false depending on whether the item is selected).

You used code like this to retrieve the selected ListItem object from an HtmlSelect control called Currency, as follows:

```
ListItem item;
item = Currency.Items(Currency.SelectedIndex);
```

If you used the ListBox web control, you can simplify this code with a clearer syntax:

```
ListItem item;
item = Currency.SelectedItem;
```

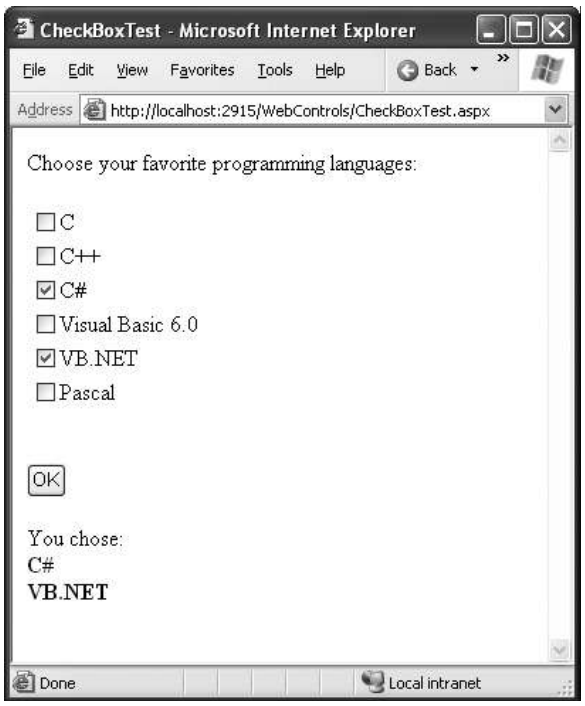
4.4.1 Multiple-Select List Controls

Some list controls can allow multiple selections. This isn't allowed for the DropDownList or RadioButtonList, but it is supported for a ListBox, provided you have set the SelectionMode property to the enumerated value ListSelectionMode.Multiple. The user can then select multiple items by holding down the Ctrl key while clicking the items in the list. With the CheckBoxList, multiple selections are always possible.

If you have a list control that supports multiple selections, you can find all the selected items by iterating through the Items collection of the list control and checking the ListItem.Selected property of each item. Figure 4.5 shows a simple web page example. It provides a list of computer languages and indicates which selections the user made when the OK button is clicked.

Notes

Figure 4.5: A Simple CheckListBox Test



The .aspx file for this page defines CheckListBox, Button, and Label controls, as shown here:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="CheckListTest.aspx.cs" Inherits="CheckListTest" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>CheckBoxTest</title>
</head>
<body>
<form runat="server">
<div>
Choose your favorite programming languages:<br /><br />
<asp:CheckBoxList ID="chklst" runat="server" /><br /><br />
<asp:Button ID="cmdOK" Text="OK" OnClick="cmdOK_Click" runat="server" />
<br /><br />
<asp:Label ID="lblResult" runat="server" />
</div>
</form>
</body>
</html>
```

The code adds items to the CheckListBox at startup and iterates through the collection when the button is clicked:

```
public partial class CheckListTest : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
```

Notes

```
{
if (!this.IsPostBack)
{
chklst.Items.Add("C");
chklst.Items.Add("C++");
chklst.Items.Add("C#");
chklst.Items.Add("Visual Basic 6.0");
chklst.Items.Add("VB.NET");
chklst.Items.Add("Pascal");
}
}
protected void cmdOK_Click(object sender, EventArgs e)
{
lblResult.Text = "You chose:<b>";
foreach (ListItem lstItem in chklst.Items)
{
if (lstItem.Selected == true)
{
// Add text to label.
lblResult.Text += "<br />" + lstItem.Text;
}
}
lblResult.Text += "</b>";
}
}
```

4.4.2 The BulletedList Control

Table 4.4: Added BulletedList Properties

Property	Description
BulletStyle	Determines the type of list. Choose from Numbered (1, 2, 3, . . .), LowerAlpha (a, b, c, . . .) and UpperAlpha (A, B, C, . . .), LowerRoman (i, ii, iii, . . .) and UpperRoman (I, II, III, . . .), and the bullet symbols Disc, Circle, Square, or CustomImage (in which case you must set the BulletImageUrl property).
BulletImageUrl	If the BulletStyle is set to CustomImage, this points to the image that is placed to the left of each item as a bullet.
FirstBulletNumber	In an ordered list (using the Numbered, LowerAlpha, UpperAlpha, LowerRoman, and UpperRoman styles), this sets the first value. For example, if you set FirstBulletNumber to 3, the list might read 3, 4, 5 (for Numbered) or C, D, E (for UpperAlpha).
Display Mode	Determines whether the text of each item is rendered as text (use Text, the default) or a hyperlink (use LinkButton or HyperLink). The difference between LinkButton and HyperLink is how they treat clicks. When you use LinkButton, the BulletedList fires a Click event that you can react to on the server to perform the navigation. When you use HyperLink, the BulletedList doesn't fire the Click event—instead, it treats the text of each list item as a relative or absolute URL, and renders them as ordinary HTML hyperlinks. When the user clicks an item, the browser attempts to navigate to that URL.

Notes

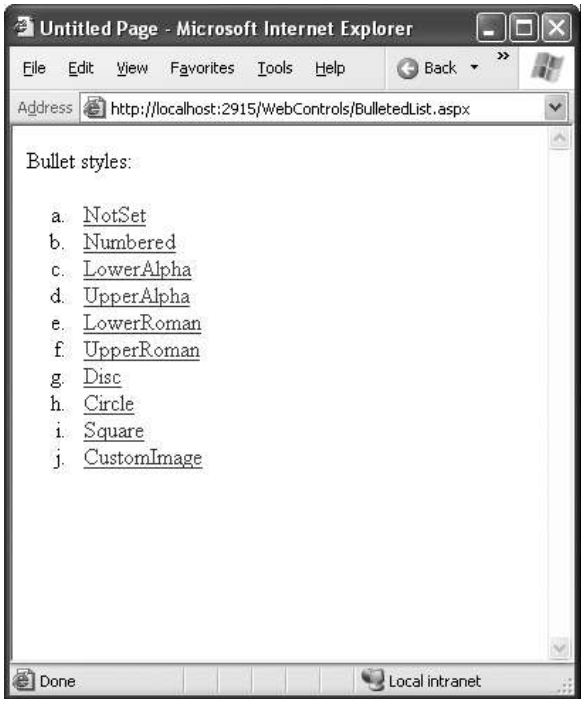
The BulletedList control is a server-side equivalent of the <ul> (unordered list) and <ol> (ordered list) elements. As with all list controls, you set the collection of items that should be displayed through the Items property. Additionally, you can use the properties in Table 4.4 to configure how the items are displayed.

If you set the DisplayMode to LinkButton, you can react to the Button.Click event to determine which item was clicked. Here's an example:

```
protected void BulletedList1_Click(object sender, BulletedListEventArgs e)
{
string itemText = BulletedList1.Items[e.Index].Text;
Label1.Text = "You choose item" + itemText;
}
```

Figure 4.6 shows all the BulletStyle values that the BulletList supports. When you click one of the items, the list changes to use that BulletStyle.

Figure 4.6: Various BulletedList Styles



4.4.3 Table Controls

Essentially, the Table control is built out of a hierarchy of objects. Each Table object contains one or more TableRow objects. In turn, each TableRow object contains one or more TableCell objects. Each TableCell object contains other ASP.NET controls of HTML content that displays information. If you're familiar with the HTML table tags, this relationship (Figure 4.7) will seem fairly logical.

To create a table dynamically, you follow the same philosophy as you would for any other web control. First, you create and configure the necessary ASP.NET objects. Then, ASP.NET converts these objects to their final HTML representation before the page is sent to the client.

Notes

Consider the example shown in Figure 4.8. It allows the user to specify a number of rows and columns as well as whether cells should have borders.

Figure 4.7: Table Control Containment

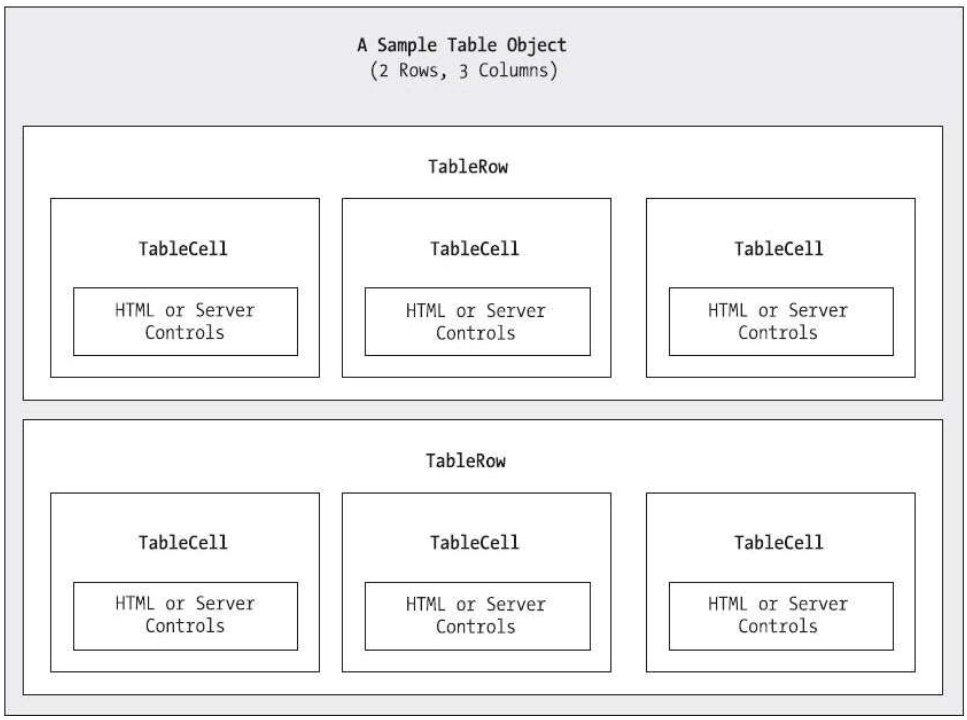
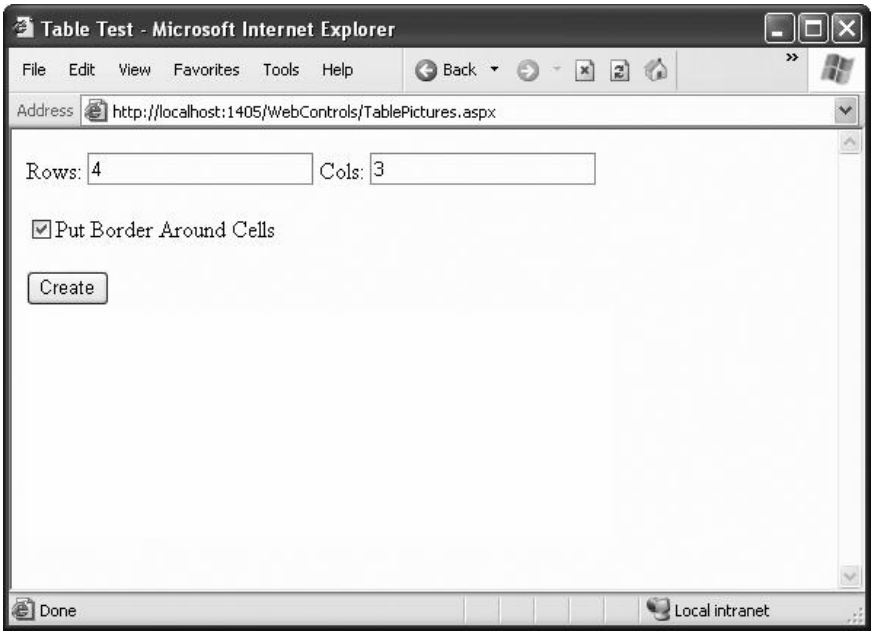


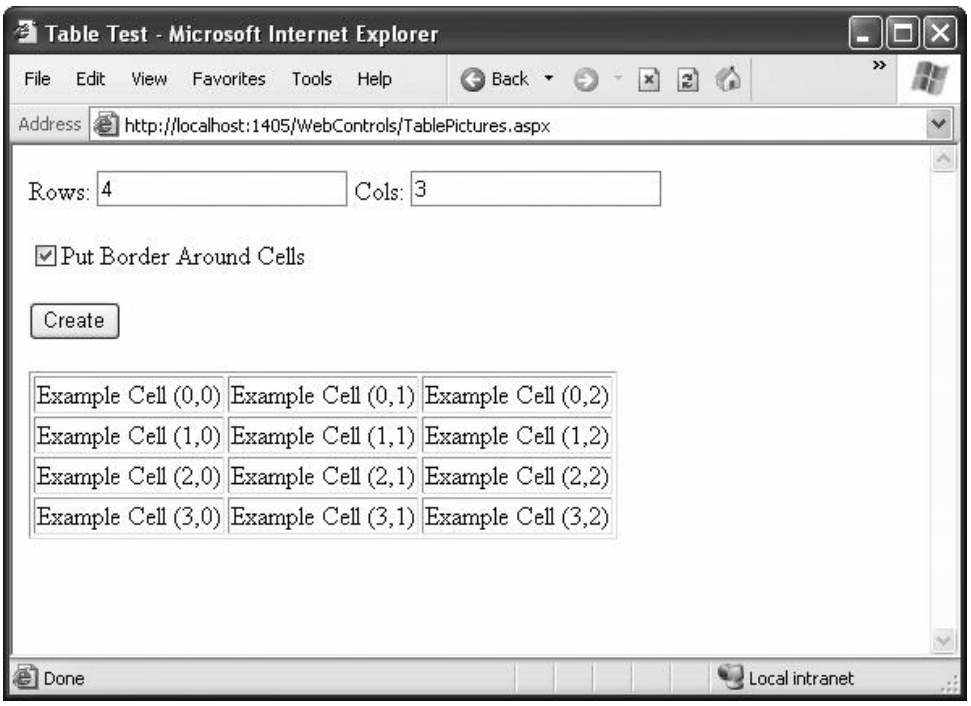
Figure 4.8: The Table Test Options



When the user clicks the Create button, the table is filled dynamically with sample data according to the selected options, as shown in Figure 4.9.

Notes

Figure 4.9: A Dynamically Generated Table



The .aspx code creates the TextBox, CheckBox, Button, and Table controls:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="TableTest.aspx.cs" Inherits="TableTest" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Table Test</title>
</head>
<body>
<form runat="server">
<div>
Rows:
<asp:TextBox ID="txtRows" runat="server" />
 Cols:
<asp:TextBox ID="txtCols" runat="server" />
<br /><br />
<asp:CheckBox ID="chkBorder" runat="server"
Text="Put Border Around Cells" />
<br /><br />
<asp:Button ID="cmdCreate" OnClick="cmdCreate_Click" runat="server"
Text="Create" />
<br /><br />
<asp:Table ID="tbl" runat="server" />
</div>
</form>
```

Notes

```
</body>
</html>
```

You'll notice that the Table control doesn't contain any actual rows or cells. To make a valid table, you would need to nest several layers of tags. The following example creates a table with a single cell that contains the text A Test Row:

```
<asp:Table ID="tbl" runat="server">
<asp:TableRow ID="row" runat="server">
<asp:TableCell ID="cell" runat="server">A Sample Value</asp:TableCell>
</asp:TableRow>
</asp:Table>
```

The table test web page doesn't have any nested elements. This means the table will be created as a server-side control object, but unless the code adds rows and cells, the table will not be rendered in the final HTML page.

The TableTest class uses two event handlers. When the page is first loaded, it adds a border around the table. When the button is clicked, it dynamically creates the required TableRow and TableCell objects in a loop.

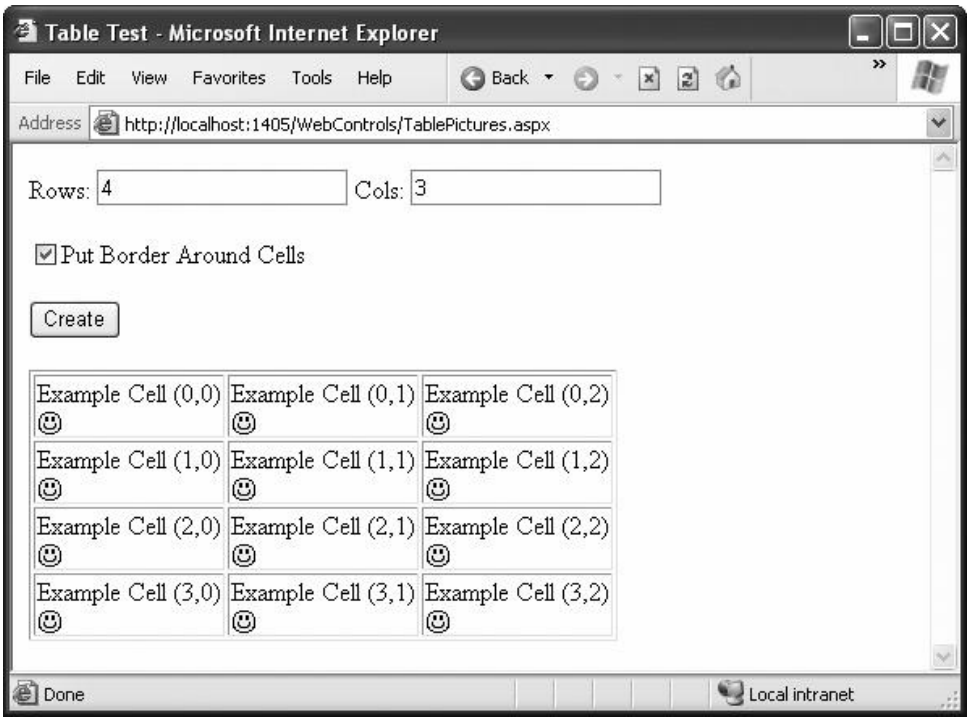
```
public partial class TableTest : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
// Configure the table's appearance.
// This could also be performed in the .aspx file
// or in the cmdCreate_Click event handler.
tbl.BorderStyle = BorderStyle.Inset;
tbl.BorderWidth = Unit.Pixel(1);
}
protected void cmdCreate_Click(object sender, EventArgs e)
{
// Remove all the current rows and cells.
// This is not necessary if EnableViewState is set to false.
tbl.Controls.Clear();
int rows = Int32.Parse(txtRows.Text);
int cols = Int32.Parse(txtCols.Text);
for (int row = 0; row < rows; row++)
{
// Create a new TableRow object.
TableRow rowNew = new TableRow();
// Put the TableRow in the Table.
tbl.Controls.Add(rowNew);
for (int col = 0; col < cols; col++)
{
// Create a new TableCell object.
TableCell cellNew = new TableCell();
cellNew.Text = "Example Cell (" + row.ToString() + ",";
cellNew.Text += col.ToString() + ")";
```

Notes

```
if (chkBorder.Checked)
{
cellNew.BorderStyle = BorderStyle.Inset;
cellNew.BorderWidth = Unit.Pixel(1);
}
// Put the TableCell in the TableRow.
rowNew.Controls.Add(cellNew);
}
}
}
}
```

This code uses the Controls collection to add child controls. Every container control provides this property. You could also use the TableCell.Controls collection to add web controls to each TableCell. For example, you could place an Image control and a Label control in each cell. In this case, you can't set the TableCell.Text property. The following code snippet uses this technique, and Figure 4.10 displays the results:

Figure 4.10: A Table with Contained Controls




```
// Create a new TableCell object.
cellNew = new TableCell();

// Create a new Label object.
Label lblNew = new Label();
```

Notes

```
lblNew.Text = "Example Cell (" + row.ToString() + "," + col.ToString() +
")<br />";
System.Web.UI.WebControls.Image imgNew = new
System.Web.UI.WebControls.Image();
imgNew.ImageUrl = "cellpic.png";

// Put the label and picture in the cell.
cellNew.Controls.Add(lblNew);
cellNew.Controls.Add(imgNew);

// Put the TableCell in the TableRow.
rowNew.Controls.Add(cellNew);
```

The real flexibility of the table test page is that each Table, TableRow, and TableCell is a full-featured object. If you want, you can give each cell a different border style, border color, and text color by setting the corresponding properties.

4.5 Web Control Events and Autopostback

The main limitations of HTML server controls is their limited set of useful events they have exactly two. HTML controls that trigger a postback, such as buttons, raise a ServerClick event. Input controls provide a ServerChange event that doesn't actually fire until the page is posted back.

Server controls are really an ingenious illusion. You'll recall that the code in an ASP.NET page is processed on the server. It's then sent to the user as ordinary HTML. Figure 4.11 illustrates the order of events in page processing.

This is the same in ASP.NET as it was in traditional ASP programming. The question is, how can you write server code that will react immediately to an event that occurs on the client?

Some events, such as the Click event of a button, do occur immediately. That's because when clicked, the button posts back the page. This is a basic convention of HTML forms. However, other actions do cause events but don't trigger a postback. An example is when the user changes the text in a text box (which triggers the TextChanged event) or chooses a new item in a list (the SelectedIndexChanged event). You might want to respond to these events, but without a postback your code has no way to run.

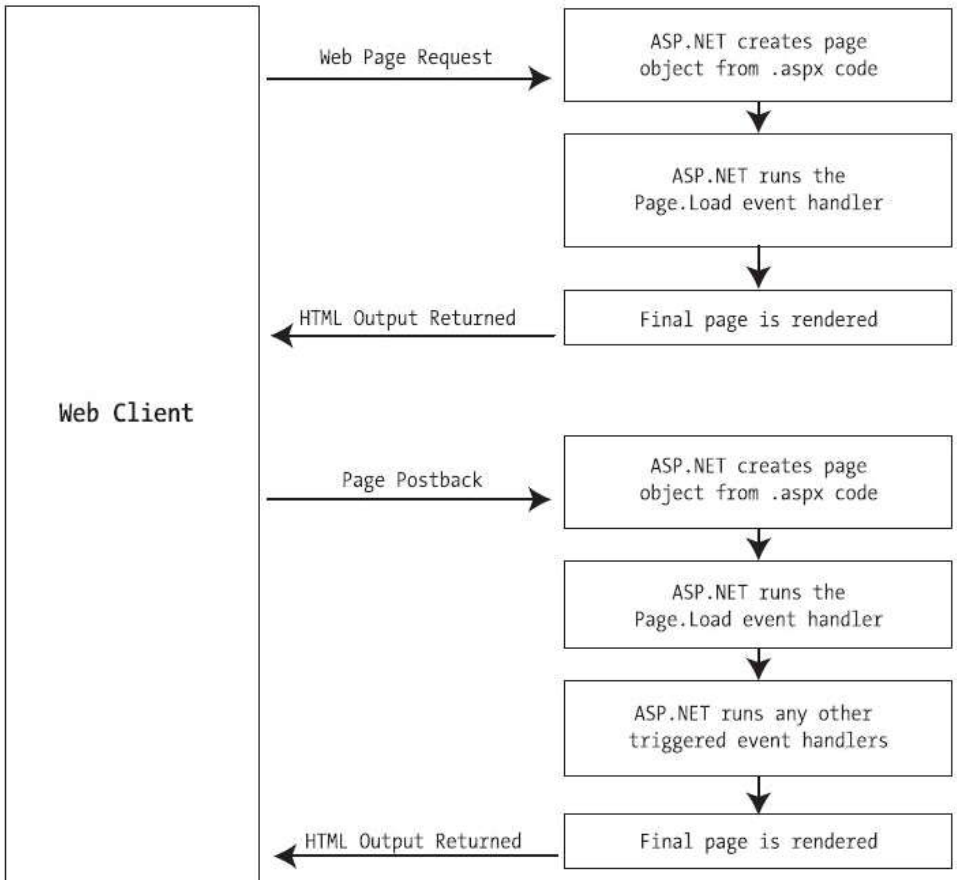
ASP.NET handles this by giving you two options:

1. You can wait until the next postback to react to the event. For example, imagine you want to react to the SelectedIndexChanged event in a list. If the user selects an item in a list, nothing happens immediately. However, if the user then clicks a button to post back the page, two events fire: Button.Click followed by TextBox.TextChanged. And if you have several controls, it's quite possible for a single postback to result in several change events, which fire one after the other, in an undetermined order.
2. You can use the automatic postback feature to force a control to post back the page immediately when it detects a specific user action. In this scenario, when the user clicks a new item in the list, the page is posted back, your code executes, and a new version of the page is returned.

Notes

The option you choose depends on the result you want. If you need to react immediately (for example, you want to update another control when a specific action takes place), you need to use automatic postbacks. On the other hand, automatic postbacks can sometimes make the page less responsive, because each postback and page refresh adds a short, but noticeable, delay and page refresh.

Figure 4.11: The Page Processing Sequence



All input web controls support automatic postbacks. Table 4.5 provides a basic list of web controls and their events.

Table 4.5: Web Control Events

Event	Web Controls that provide it	Always Posts Back
Click	Button, ImageButton	True
TextChanged	TextBox (fires only after the user changes the focus to another control)	False
CheckedChanged	CheckBox, RadioButton	False
SelectedIndexChanged	DropDownList, ListBox, CheckBoxList, RadioButtonList	False

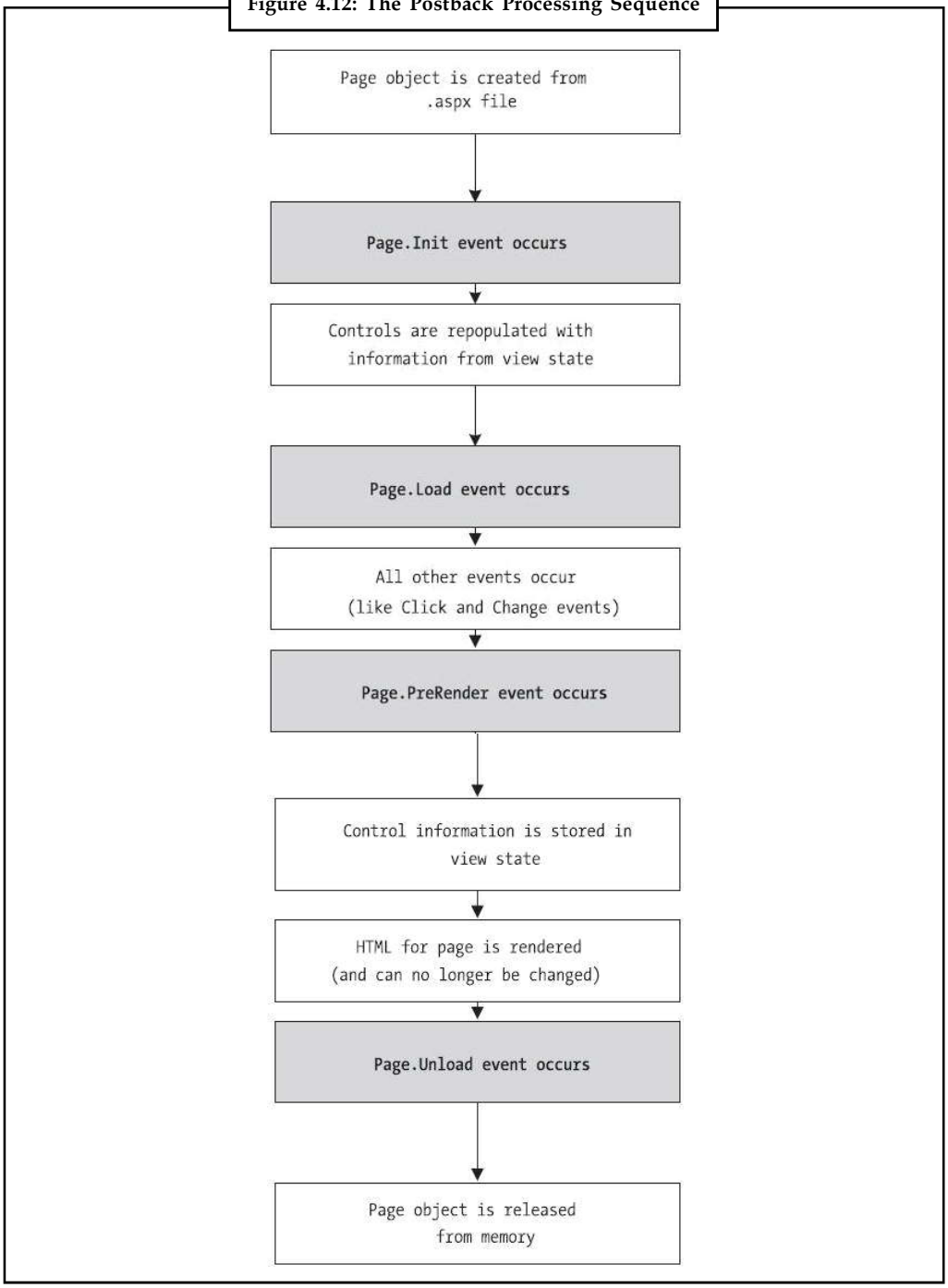
If you want to capture a change event (such as TextChanged, CheckedChanged, or SelectedIndexChanged) immediately, you need to set the control's AutoPostBack property to true. This way, the page will be submitted automatically when the user interacts with the

Notes

control (for example, picks a selection in the list, clicks a radio button or a check box, or changes the text in a text box and then tabs away to a new control).

When the page is posted back, ASP.NET will examine the page, load all the current information, and then allow your code to perform some extra processing before returning the page back to the user (Figure 4.12). Depending on the result you want, you could have a page that has some controls that post back automatically and others that don't.

Figure 4.12: The Postback Processing Sequence



Notes

This postback system isn't ideal for all events. For example, some events that you may be familiar with from Windows programs, such as mouse movement events or key press events, aren't practical in an ASP.NET application. Resubmitting the page every time a key is pressed or the mouse is moved would make the application unbearably slow and unresponsive.

How Postback Events Work?

One common example of client-side web programming is JavaScript, which uses simple code that's limited in scope and is executed by the browser. ASP.NET uses the client-side abilities of JavaScript to bridge the gap between client-side and server-side code. (Another scripting language is VBScript, but JavaScript is the only one that works on all modern browsers, including Internet Explorer, Firefox, Opera, Safari, and Netscape.)

Here's how it works: If you create a web page that includes one or more web controls that are configured to use AutoPostBack, ASP.NET adds a special JavaScript function to the rendered HTML page. This function is named __doPostBack(). When called, it triggers a postback, sending data back to the web server.

ASP.NET also adds two additional hidden input fields that are used to pass information back to the server. This information consists of the ID of the control that raised the event and any additional information that might be relevant. These fields are initially empty, as shown here:

```
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value
="" />
```

The __doPostBack() function has the responsibility for setting these values with the appropriate information about the event and then submitting the form. A slightly simplified version of the __doPostBack() function is shown here:

```
<script language="text/javascript">
<!—
function __doPostBack(eventTarget, eventArgument) {
var theform = document.Form1;
theform.__EVENTTARGET.value = eventTarget;
theform.__EVENTARGUMENT.value = eventArgument;
theform.submit();
}
// —>
</script>
```

Remember, ASP.NET generates the __doPostBack() function automatically, provided at least one control on the page uses automatic postbacks.

Finally, any control that has its AutoPostBack property set to true is connected to the __doPostBack() function using the onclick or onchange attributes. These attributes indicate what action the browser should take in response to the client-side JavaScript events onclick and onchange.

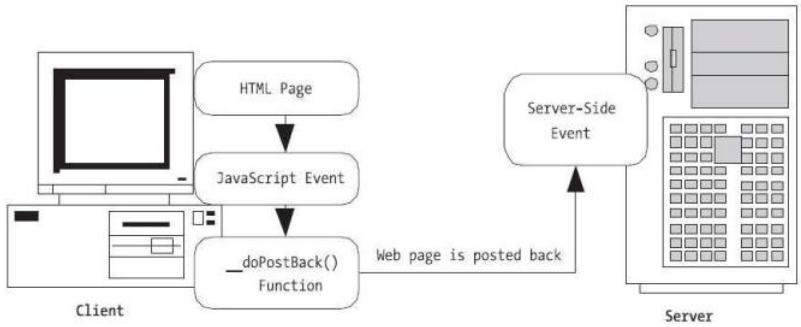
The following example shows the tag for a list control named lstBackColor, which posts back automatically. Whenever the user changes the selection in the list, the client-side onchange event fires. The browser then calls the __doPostBack() function, which sends the page back to the server.

```
<select ID="lstBackColor" onchange="__doPostBack('lstBackColor','')"
language="javascript">
```

Notes

In other words, ASP.NET automatically changes a client-side JavaScript event into a server-side ASP.NET event, using the __doPostBack() function as an intermediary. Figure 4.13 shows this process.

Figure 4.13: An Automatic Postback



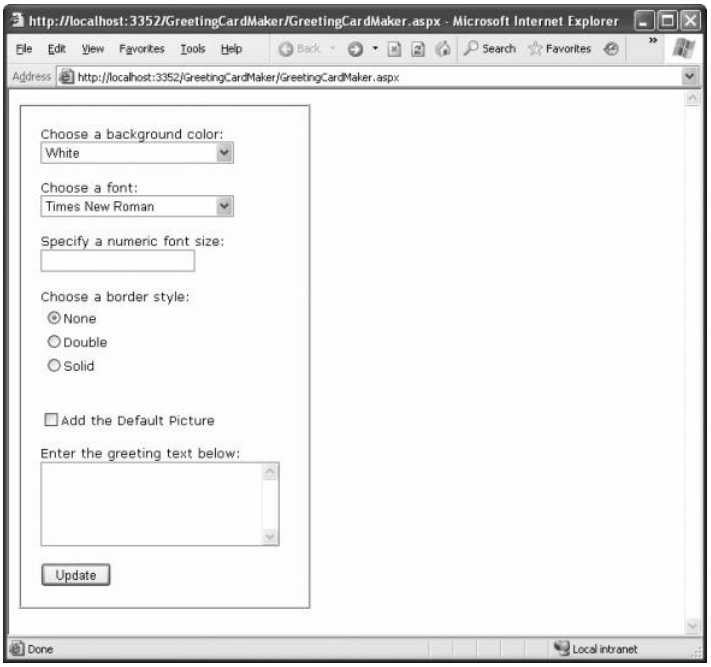
Task

Build a web page that will the user made some type of input into a textbox before exiting.

4.6 A Simple Web Page

Now that you've had a whirlwind tour of the basic web control model, it's time to put it to work with the second single-page utility. In this case, it's a simple example for a dynamic e-card generator. You could extend this sample (for example, allowing users to store e-cards to the database), but even on its own, this example demonstrates basic control manipulation with ASP.NET.

Figure 4.14: The e-card Generator



Notes

The web page is divided into two regions. On the left is an ordinary tag containing a set of web controls for specifying card options. On the right is a Panel control (named pnlCard), which contains two other controls (lblGreeting and imgDefault) that are used to display userconfigurable text and a picture. This text and picture represents the greeting card. When the page first loads, the card hasn't yet been generated, and the right portion is blank (Figure 4.14).

Whenever the user clicks the Update button, the page is posted back and the "card" is updated (Figure 4.15).

Figure 4.15: A User-configured Greeting Card

The .aspx layout code is straightforward. Of course, the sheer length of it makes it difficult to work with efficiently. Here's the markup, without the formatting details for the element:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="GreetingCardMaker.aspx.cs" Inherits="GreetingCardMaker" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Greeting Card Maker</title>
</head>
<body>
<form runat="server">
<div>
```

Notes

```
<!— Here are the controls: —>
Choose a background color:<br />
<asp:DropDownList ID=”lstBackColor” runat=”server” Width=”194px”
Height=”22px”/><br /><br />
Choose a font:<br />
<asp:DropDownList ID=”lstFontName” runat=”server” Width=”194px”
Height=”22px” /><br /><br />
Specify a numeric font size:<br />
<asp:TextBox ID=”txtFontSize” runat=”server” /><br /><br />
Choose a border style:<br />
<asp:RadioButtonList ID=”lstBorder” runat=”server” Width=”177px”
Height=”59px” /><br /><br />
<asp:CheckBox ID=”chkPicture” runat=”server”
Text=”Add the Default Picture”></asp:CheckBox><br /><br />
Enter the greeting text below:<br />
<asp:TextBox ID=”txtGreeting” runat=”server” Width=”240px” Height=”85px”
TextMode=”MultiLine” /><br /><br />
<asp:Button ID=”cmdUpdate” OnClick=”cmdUpdate_Click”
runat=”server” Width=”71px” Height=”24px” Text=”Update” />
</div>

<!— Here is the card: —>

<asp:Panel ID=”pnlCard” runat=”server”
Width=”339px” Height=”481px”
HorizontalAlign=”Center”><br /> 
<asp:Label ID=”lblGreeting” runat=”server” Width=”256px”
Height=”150px” /><br /><br /><br />
<asp:Image ID=”imgDefault” runat=”server” Width=”212px”
Height=”160px” />
</asp:Panel>
</form>
</body>
</html>
```

The code follows the familiar pattern with an emphasis on two events: the Page.Load event, where initial values are set, and the Button.Click event, where the card is generated.

```
public partial class GreetingCardMaker : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
if (!this.IsPostBack)
{
// Set color options.
lstBackColor.Items.Add(“White”);
```

Notes

```
lstBackColor.Items.Add("Red");
lstBackColor.Items.Add("Green");
lstBackColor.Items.Add("Blue");
lstBackColor.Items.Add("Yellow");
// Set font options.
lstFontName.Items.Add("Times New Roman");
lstFontName.Items.Add("Arial");
lstFontName.Items.Add("Verdana");
lstFontName.Items.Add("Tahoma");
// Set border style options by adding a series of
// ListItem objects.
ListItem item = new ListItem();
// The item text indicates the name of the option.
item.Text = BorderStyle.None.ToString();
// The item value records the corresponding integer
// from the enumeration. To obtain this value, you
// must cast the enumeration value to an integer,
// and then convert the number to a string so it
// can be placed in the HTML page.
item.Value = ((int)BorderStyle.None).ToString();

// Add the item.
lstBorder.Items.Add(item);

// Now repeat the process for two other border styles.
item = new ListItem();
item.Text = BorderStyle.Double.ToString();
item.Value = ((int)BorderStyle.Double).ToString();
lstBorder.Items.Add(item);
item = new ListItem();
item.Text = BorderStyle.Solid.ToString();
item.Value = ((int)BorderStyle.Solid).ToString();
lstBorder.Items.Add(item);

// Select the first border option.
lstBorder.SelectedIndex = 0;

// Set the picture.
imgDefault.ImageUrl = "defaultpic.png";
}
}
protected void cmdUpdate_Click(object sender, EventArgs e)
{
```

Notes

```
// Update the color.
pnlCard.BackColor = Color.FromName(lstBackColor.SelectedItem.Text);

// Update the font.
lblGreeting.Font.Name = lstFontName.SelectedItem.Text;
if (Int32.Parse(txtFontSize.Text) > 0)
{
lblGreeting.Font.Size =
FontUnit.Point(Int32.Parse(txtFontSize.Text));
}
// Update the border style. This requires two conversion steps.
// First, the value of the list item is converted from a string
// into an integer. Next, the integer is converted to a value in
// the BorderStyle enumeration.
int borderValue = Int32.Parse(lstBorder.SelectedItem.Value);
pnlCard.BorderStyle = (BorderStyle)borderValue;
// Update the picture.
if (chkPicture.Checked)
{
imgDefault.Visible = true;
}
else
{
imgDefault.Visible = false;
}
// Set the text.
lblGreeting.Text = txtGreeting.Text;
}
}
```

As you can see, this example limits the user to a few preset font and color choices. The code for the BorderStyle option is particularly interesting. The lstBorder control has a list that displays the text name of one of the BorderStyle enumerated values. You'll remember from the introductory unit s that every enumerated value is really an integer with a name assigned to it. The lstBorder also secretly stores the corresponding number so that the code can retrieve the number and set the enumeration easily when the user makes a selection and the cmdUpdate_Click event handler fires.

Task

Build a series of three check so that when one is checked, the other two will be unchecked. The functionality should be the same as three radio buttons.

Notes

Case Study

Online Banking Solution Benefits after ASP.NET MVC Replaces Ruby on Rails

Jwaala, creator of the MoneyTracker online banking solution, prides itself in helping to change the face of Internet banking with its personal finance management innovations. The company, based in Austin, Texas, chose as its name the Hindi word for passion. A passion for online banking and personal financial management is seen throughout MoneyTracker, which includes integrated account aggregation, natural language searching, personal financial management, consumer remote deposit, secure document repository, widgets, and infinitely customizable alerting options.

Banks and credit unions across the United States deploy MoneyTracker to enable their users to add a layer of intuitive, easy-to-use online banking and personal financial management tools on top of their financial institution's core banking infrastructure. Users enjoy the single dashboard-like controls of MoneyTracker, which behind the scenes is uniting separate, and often disparate, applications and data stores to provide users with a unified online banking experience.

Jwaala created MoneyTracker using the Ruby on Rails development environment, and designed it for deployment on the Linux operating system, Apache Web server, and the MySQL database platform (sometimes referred to as the LAMR stack).

Developers at Jwaala liked working with Ruby on Rails because it provides a framework and tools for creating Web-based applications, and because of its use of the Model-View-Controller (MVC) architecture that simplifies application programming. However, the company found that many of its potential customers in the financial industry had limited experience about deploying an application that required the LAMR stack.

"The combination of Ruby on Rails and Linux, Apache, and MySQL had served us well," says Andrew Taylor, Chief Executive Officer and Chief Technical Officer of Jwaala. "But by early 2008 we were seeing that more of our customers in the financial industry preferred the Windows Server operating system. To better match their needs, we decided to look into moving our application to the Microsoft Application Platform, including Microsoft SQL Server 2008."

Solution:

Before making the move, the company needed to determine whether the development tools of the Microsoft Application Platform—Visual Studio 2008 and the Microsoft .NET Framework 3.5 could meet the needs of Jwaala developers as well as had Ruby on Rails.

"About the time we started looking at the Microsoft Application Platform, we saw two new developments that simplified our decision to move away from Ruby on Rails and the LAMR stack," Taylor says. "Microsoft released its ASP.NET MVC [a downloadable addition to the Microsoft .NET Framework 3.5] which provided a very similar development framework to what we were used to using with Ruby on Rails; and we discovered Microsoft LINQ [Language Integrated Query, a Microsoft .NET Framework component] that provides native data querying capabilities. The combination of ASP.NET MVC and LINQ really made it easy for us to make the switch."

As part of its due diligence, Jwaala chose some representative components from its existing Money Tracker solution and rewrote them using ASP.NET MVC and LINQ. "The code

Contd...

Notes

conversion went easily and we felt as if what we ended up with was in some ways even better than what we had before," Taylor notes. "So we began the process of completely rewriting all of our code. We were leaving Linux, Apache, MySQL, and Ruby on Rails. The effort took about four months, and we feel as if we are now much better positioned for the future."

Deployment scenarios differ according to customer preference and existing infrastructure, but the recommended deployment architecture includes:

1. ***Presentation Tier:*** Customers access MoneyTracker using a browser, avoiding the need for downloading client-side applications. The presentation tier uses Window Server 2008 Internet Information Services (IIS) 7.0 Web server technology (the solution also supports Windows Server 2003). The presentation tier is hosted on a Web farm using Windows Server 2008 Network Load Balancing to dynamically distribute traffic across server computers.
2. ***Application Tier:*** MoneyTracker is deployed on a dedicated platform using two server computers configured as a 2-node cluster. MoneyTracker seamlessly interoperates with the financial institution's core banking applications, which are frequently also deployed on the Microsoft Application Platform, though MoneyTracker can interface with core systems deployed using other solution stacks. The application works by passing user instructions to the core banking system and confirming results as transactions or queries are completed on the core system. The MoneyTracker application was created using Visual Studio 2008, the Microsoft .NET Framework 3.5, ASP.NET MVC, and LINQ.
3. ***Database Tier:*** MoneyTracker stores records of user interactions with the core system (but not the actual core data) using a dedicated relational database hosted on Microsoft SQL Server 2008 (it also supports SQL Server 2005).
4. ***Reporting Tier:*** Currently Jwaala exposes interfaces for customers to use in connecting their own reporting solutions, though the company plans to take advantage of SQL Server 2008 Reporting Services and SQL Server 2008 Analysis Services to enable financial institutions to create recurring and ad hoc reports and to generate analytics to better monitor and refine their service delivery.

Benefits

Moving its MoneyTracker application from Ruby on Rails and the rest of the LAMR stack to code created using ASP.NET MVC and LINQ for deployment on the Microsoft Application Platform has given Jwaala the easier sales cycle it had hoped for and faster deployments at customer sites. The company has seen a reduced need for customer support because its banking and credit union customers already are familiar with Windows Server and the Microsoft Application Platform. Jwaala has benefited from the efficient development environment it has found with Microsoft development tools, and ease of interoperability with backend infrastructure.

Easier Sales Cycle

Jwaala has found that its sales cycle is easier and faster since migrating its development and deployment infrastructure to the Microsoft Application Platform.

"The move from Linux and MySQL to Windows Server and SQL Server has greatly smoothed out the sales cycle for us," says Kelly Dowell, Chief Operating Officer at Jwaala. "Our customers are financial institutions, which means they are already well acquainted with the Microsoft Application Platform and reluctant to introduce another technology such as

Contd...

Notes

Linux and MySQL into their enterprise environment. Our sales people are reporting back to us that sales are closing easier and more quickly simply because they no longer have to make the case about why an organization should deploy Linux and MySQL to support our application. Removing that conversation really speeds things up."

The enterprise-grade performance of SQL Server and Windows Server made it easy for Jwaala to migrate its existing customer base from a LAMR stack to the Microsoft Application Platform.

"All but two of our existing customers have already migrated from Linux, Apache and MySQL to our new solution, and the final two will be migrated by the end of the year," says Dowell. "The response from existing customers was very positive. We went to them and said, 'This is a decision we've made, and here's why. Let's put together a plan to migrate you over to the new environment.' Our assumption was that banks and credit unions would be more comfortable with SQL Server and Windows Server than with MySQL and Linux, and that has proven to be true."

Faster Deployment

Solution deployments are easier to accomplish and go faster since moving MoneyTracker from the LAMR stack to the Microsoft Application Platform because customers are already familiar with the operating system, database, and the other infrastructure that the solution is being deployed to.

"Moving MoneyTracker to the Microsoft Application Platform places the responsibility for the first part of the deployment cycle onto the customer—which is the way they prefer to have it," Taylor says. "Each organization has its own procedures for preparing new server computers for deployment to their Windows-based infrastructure. They can set up the operating system and everything else to their own criteria, and then we load our solution."

This is a pleasant change for Jwaala and its customers. "When our solution was based on Linux, Apache, and MySQL, we usually had to do the entire initial infrastructure configuration for them because they weren't used to supporting these technologies," Taylor says. "This was fine with us, but the customer would feel uncomfortable having to support an operating system, database, and other technology they weren't accustomed to. Deploying MoneyTracker on the Microsoft Application Platform makes life easier for us and for them."

Reduced Need for Customer Support

Jwaala and its customers benefit from the easy systems administration that comes with deploying MoneyTracker on the Microsoft Application Platform. The company immediately noticed it had fewer systems administration calls from customers because they were already used to working with Windows Server, SQL Server, and other elements of the Microsoft Application Platform.

"Now our customers can self-support their MoneyTracker deployment, but this often wasn't the case when we were deploying a Ruby on Rails application on a platform of Linux, Apache, and MySQL," Taylor says. "We used to have to do a lot of platform troubleshooting with our LAMR stack. Customers didn't know how to do simple things like database backups, certificate renewals, and operating system monitoring. There's a whole class of support-related issues that we don't get involved with anymore because the customers already know how to work with the Microsoft Application Platform."

Contd...

Notes

The familiarity that Jwaala customers have with the Microsoft Application Platform is a big relief to all parties involved. "When we were deploying our solution on Linux, Apache, and MySQL, some of our customers acted as if we were placing a spaceship in their IT center," Taylor says. "They were afraid to touch it. Every minute that we used to spend helping someone with routine IT functions in the Linux environment was a minute we didn't spend enhancing, developing, deploying, supporting, and selling our product."

Efficient Development Environment

ASP.NET MVC and LINQ were enabling factors for Jwaala. "We couldn't have switched deployment platforms without a development environment that was at least as good as Ruby on Rails," Taylor says. "We couldn't have done this without ASP.NET MVC and LINQ, which have proven to be equal to and in some areas superior to what we could do with Ruby on Rails."

"Every developer on our team has had extensive experience with all of the major environments, because without that experience you can't appreciate what Rails did," Taylor says. "Java has had MVC Web-based environments for a long time, for example. But when you say MVC there's MVC that's hard to work with and MVC that is smooth and productive. With that background we respected Ruby on Rails and fully appreciated the efficiency of ASP.NET MVC."

The Jwaala development team found that in addition to an excellent MVC implementation, they gained a number of development tools that were either absent from Ruby on Rails, or present but not as well developed.

"For the most part, Ruby is extremely loosely typed, which has benefits but these benefits start to fade away as your code base grows," Taylor says. "You need the strong typing provided by LINQ and other parts of the platform. When your code base gets larger, there's just more and more to remember. Without the kind of features Visual Studio provides—such as IntelliSense technology for automatic code completion, compile-time checking, and a powerful debugger—your project can get bogged down. Ruby on Rails has debugging, but nothing that is as easy to use and as proactive as what Visual Studio has. Visual Studio is a powerful development environment that can really boost productivity."

In addition, Jwaala is very open with their technology, to the point where about 25 percent of their customers license the actual source code. All of these customers are already experienced Windows and .NET shops.

Jwaala developers also like the visibility into ASP.NET MVC code that Microsoft provides. "Microsoft makes its ASP.NET MVC source code available so developers actually can see and debug into the framework that they're relying on," Taylor says. "This is important to us because when you're building a Web-based application, the framework you're riding on is about as important as it gets. The ability to look inside the code to see what's going on helps you to figure things out."

Ease of Interoperability

Rewriting the MoneyTracker application code for deployment on the Microsoft Application Platform has made it easier for Jwaala to interoperate with the backend infrastructure of its financial services customers to create optimal solutions.

"Our solution blends online banking with personal finance management so that when a user logs into his or her bank or credit union, instead of just seeing a list of transactions they are provided with a rich view into—and tools to simplify—their personal finance

Contd...

Notes

management," Taylor says. "Now to get all that data into a single application for the user to work with, we have to interoperate with all sorts of other systems at the bank or credit union transaction processing systems, credit card systems, document management systems. These applications and data stores tend to be all over the place. Some information might be coming from a UNIX server or from any number of database management systems. We've found it is a lot easier to communicate with these other systems using the .NET Framework and the Microsoft Application Platform than it is when working from within a Linux environment."

4.7 Summary

- Topics common to all ASP.NET server controls were covered, such as events, syntax, programmatic access to controls during runtime (using the ID property), and the use of VS2005 to build your web site using controls.
- However, it did not go into significant detail about any specific controls.
- This unit provides a wealth of detail about many of the basic ASP.NET controls, including the TextBox, Button, CheckBox, and RadioButton controls, lists, tables, and images.
- It discusses the features and properties common to many controls and surveys the specific details of the basic ASP.NET server controls included with the .NET Framework.

4.8 Keywords

Web Forms: The ASP.NET page framework, which consists of programmable Web pages (called Web Forms pages) that contain reusable server controls.

Web Parts Controls: A way of referring generally to any of the various types of controls in the Web Parts control set.

Web Server Control: An ASP.NET server control that belongs to the System.Web.UI.WebControls namespace. Web server controls are richer and more abstract than HTML server controls. A Web server control has an asp tag prefix on an ASP.NET page, such as <asp:Button runat ="server" />.

Web Service: An application hosted on a Web server that provides information and services to other network applications using the HTTP and XML protocols. A Web service is conceptually an URL-addressable library of functionality that is completely independent of the consumer and stateless in its operation.

4.9 Self Assessment

Fill in the blanks:

1. ..... emphasize the future of web design.
2. A ..... is programmed as an object but doesn't necessarily correspond to a single element in the final HTML page.
3. Web control ..... are defined in the System.Web.UI.WebControls namespace.
4. .....are used heavily in the .NET class library to group a set of related constants.
5. The Font property actually references a full FontInfo object, which is defined in the .....
6. The list controls include the ListBox, DropDownList, CheckBoxList, RadioButtonList, and .....

Notes

State whether the following statements are True or False:

7. ASP.NET uses the client-side abilities of JavaScript to bridge the gap between client-side and server-side code.
8. ASP.NET tags have a special format. They always begin with the prefix asp: followed by the class name.
9. Use Percentage() to supply a value in pixels.
10. The BulletedList control is a server-side equivalent of the <ul> (unordered list) and <ol> (ordered list) elements.

4.10 Review Questions

1. What is a postback?
2. What are the two types of postbacks in ASP.NET? What is the difference between them?
3. What property is found on every control?
4. What control would you use to have the user enter a password, but keep the text hidden?
5. What control would you use if you have a list of 20 items, and the user can select as many as they want?
6. How do you make single radio buttons mutually exclusive?
7. What can you use a Panel control for?
8. What does the SelectedItem property retrieve?
9. How do you include a control on the page, but not render it?
10. What do you do to make the target of a HyperLink control open in a new window?
11. When you're creating a web page, often knowing which controls to use is a bigger challenge than using the controls properly. That's what you're going to practice here. Imagine a page for a busy ice cream shop that lets you preorder an ice cream cone so it will be ready for you when you get to the counter. The page should have three controls. The first control asks users to select the type of ice cream from the following list: Vanilla, Chocolate, Strawberry, Mint, Butter Pecan, Coffee, Pistachio, Coconut, Bubble Gum, and Cotton Candy. Only one type of ice cream is allowed per order. The second control asks the user to select the toppings they want: chocolate sprinkles, rainbow sprinkles, hot fudge, caramel, cookie dough, Oreo cookies, pretzel bits, walnuts, coffee beans, or crushed candy bars. It's a gourmet ice cream shop, so customers can have as many toppings as they like. The third control asks users to choose a cone or a dish. Obviously, only one is allowed. Make sure to include a way for users to submit their order.

Answers: Self Assessment

1. Web controls
2. web control
3. classes
4. Enumerations
5. System.Web.UI.WebControls namespace
6. BulletedList
7. True
8. True
9. False
10. True

Notes

4.11 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Notes

Unit 5: State Management

CONTENTS

Objectives

Introduction

5.1 View State

5.1.1 The ViewState Collection

5.1.2 A View State Example

5.2 Transferring Information between Pages

5.2.1 Cross-page Posting

5.2.2 The Query String

5.3 Cookies

5.4 Session State

5.4.1 Session Tracking

5.4.2 Using Session State

5.5 Session State Configuration

5.5.1 Cookieless

5.5.2 Timeout

5.5.3 Mode

5.6 Application State

5.7 Summary

5.8 Keywords

5.9 Self Assessment

5.10 Review Questions

5.11 Further Readings

Objectives

After studying this unit, you will be able to:

- Define view state
- Know transferring information between pages
- Describe cookies
- Explain session state
- Define application state

Notes

Introduction

ASP.NET is intrinsically a server-side technology. All of the code that goes into creating a Web Form executes on the web server. The entire .NET Framework is geared toward generating markup compliant with the HTML 4.0 specification (www.w3.org). There's no facility for spawning a .NET process on the client. With ASP and COM there are ActiveX controls. With Java there are applets. With .NET nada.

Creating rich functionality using ASP.NET requires much less code than it does using ASP and other web development environments. You realize a large portion of this savings through the "state maintenance" the Framework provides. This accounts for, among other things, a drop-down list's capability to maintain the entries in the list across postbacks, all HTML input controls being able to maintain their values across postbacks, and, in more advanced cases, the capability of the DataGrid to maintain its entire HTML table when one of its events causes a postback.

The most significant difference between programming for the web and programming for the desktop is state management how you store information over the lifetime of your application. This information can be as simple as a user's name, or as complex as a stuffed-full shopping cart for an e-commerce store.

In a traditional Windows application, there's little need to think about state management. Memory is plentiful and always available, and you only need to worry about a single user. In a web application, it's a different story. Thousands of users can simultaneously run the same application on the same computer (the web server), each one communicating over a stateless HTTP connection. These conditions make it impossible to design a web application like a traditional Windows program.

5.1 View State

One of the most common ways to store information is in view state. View state uses a hidden field that ASP.NET automatically inserts in the final, rendered HTML of a web page. It's a perfect place to store information that's used for multiple postbacks in a single web page.

Web controls store most of their property values in view state, provided the control's EnableViewState property is set to true (which is the default).

However, view state isn't limited to web controls. Your web page code can add bits of information directly to the view state of the containing page and retrieve it later after the page is posted back. The type of information you can store includes simple data types and your own custom objects.

5.1.1 The ViewState Collection

The ViewState property of the page provides the current view state information. This property is an instance of the StateBag collection class. The StateBag is a dictionary collection, which means every item is stored in a separate "slot" using a unique string name.

Example: Consider this code:

```
// The this keyword refers to the current Page object. It's optional.
this.ViewState["Counter"] = 1;
```

This places the value 1 (or rather, an integer that contains the value 1) into the ViewState collection and gives it the descriptive name Counter. If currently no item has the name Counter, a new item will be added automatically. If an item is already stored under the name Counter, it will be replaced.

Notes

When retrieving a value, you use the key name. This extra step is required because the ViewState collection stores all items as basic objects, which allows it to handle many different data types.

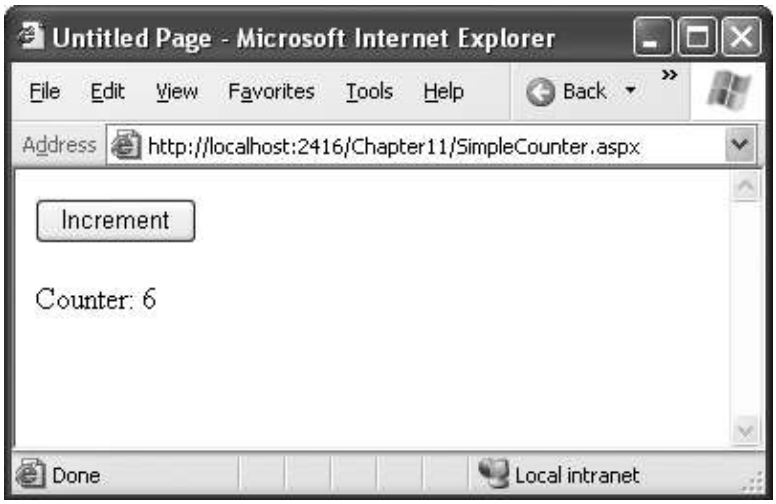
Here's the code that retrieves the counter from view state and converts it to an integer:

```
int counter;
counter = (int)this.ViewState["Counter"];
```

Notes ASP.NET provides many collections that use the same dictionary syntax. This includes the collections you'll use for session and application state, as well as those used for caching and cookies.

5.1.2 A View State Example

Figure 5.1: A Simple View State Counter



The following example is a simple counter program that records how many times a button is clicked. Without any kind of state management, the counter will be locked perpetually at 1. With careful use of view state, the counter works as expected.

```
public partial class SimpleCounter : System.Web.UI.Page
{
protected void cmdIncrement_Click(Object sender, EventArgs e)
{
int counter;
if (ViewState["Counter"] == null)
{
counter = 1;
}
else
{
counter = (int)ViewState["Counter"] + 1;
}
ViewState["Counter"] = counter;
```

Notes

```
lblCount.Text = "Counter: " + counter.ToString();
}
}
```

The code checks to make sure the item exists in view state before it attempts to retrieve it. Otherwise, you could easily run into problems such as the infamous null reference exception.

Figure 5.1 shows the output for this page.

5.2 Transferring Information between Pages

One of the most significant limitations with view state is that it's tightly bound to a specific page. If the user navigates to another page, this information is lost. This problem has several solutions, and the best approach depends on your requirements.

In the following sections, you'll learn two basic techniques to transfer information between pages: cross-page posting and the query.

5.2.1 Cross-page Posting

A cross-page postback is a technique that extends the postback mechanism you've already learned about so that one page can send the user to another page, complete with all the information for that page. This technique sounds conceptually straightforward, but it's a potential minefield. If you're not careful, it can lead you to create pages that are tightly coupled to others and difficult to enhance and debug.

The infrastructure that supports cross-page postbacks is a new property named PostBackUrl, which is defined by the IButtonControl interface and turns up in button controls such as ImageButton, LinkButton, and Button. To use cross-posting, you simply set PostBackUrl to the name of another web form. When the user clicks the button, the page will be posted to that new URL with the values from all the input controls on the current page.

Here's an example—a page named CrossPage1.aspx that defines a form with two text boxes and a button. When the button is clicked, it posts to a page named CrossPage2.aspx.

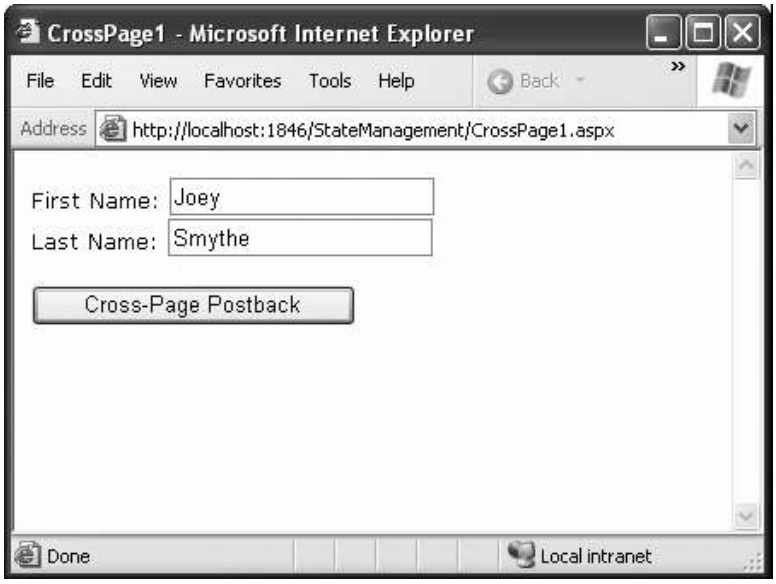
```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="CrossPage1.aspx.cs"
Inherits="CrossPage1" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>CrossPage1</title>
</head>
<body>
<form id="form1" runat="server" >
<div>
First Name:
<asp:TextBox ID="txtFirstName" runat="server"></asp:TextBox>
<br />
Last Name:
<asp:TextBox ID="txtLastName" runat="server"></asp:TextBox>
<br />
<br />
```

Notes

```
<asp:Button runat="server" ID="cmdPost"
PostBackUrl="CrossPage2.aspx" Text="Cross-Page Postback" /><br />
</div>
</form>
</body>
</html>
```

The CrossPage1 page doesn't include any code. Figure 5.2 shows how it appears in the browser.

Figure 5.2: The Source of a Cross-page Postback



Now if you load this page and click the button, the page will be posted back to CrossPage2.aspx. At this point, the CrossPage2.aspx page can interact with CrossPage1.aspx using the Page.PreviousPage property. Here's an event handler that grabs the title from the previous page and displays it:

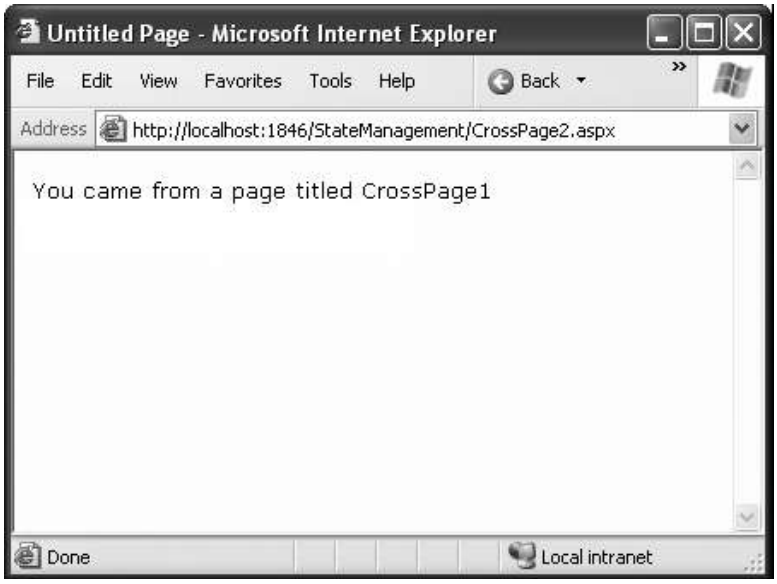
```
public partial class CrossPage2 : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
if (PreviousPage != null)
{
lblInfo.Text = "You came from a page titled " +
PreviousPage.Title;
}
}
}
```

Note that this page checks for a null reference before attempting to access the PreviousPage object. If it's a null reference, no cross-page postback took place. This means CrossPage2.aspx was requested directly, or CrossPage2.aspx posted back to itself. Either way, no PreviousPage object is available.

Notes

Figure 5.3 shows what you'll see when CrossPage1.aspx posts to CrossPage2.aspx.

Figure 5.3: The Target of a Cross-page Postback



Getting more Information from the Source Page

The previous example shows an interesting initial test, but it doesn't really allow you to transfer any useful information. After all, you're probably interested in retrieving specific details (such as the text in the text boxes of CrossPage1.aspx) from CrossPage2.aspx. The title alone isn't very interesting.

To get more specific details, such as control values, you need to cast the PreviousPage reference to the appropriate page class (in this case it's the CrossPage1 class). Here's an example that handles this situation properly, by checking first whether the PreviousPage object is an instance of the expected class:

```
protected void Page_Load(object sender, EventArgs e)
{
CrossPage1 prevPage = PreviousPage as CrossPage1;
if (prevPage != null)
{
// (Read some information from the previous page.)
}
}
```

You can also solve this problem in another way. Rather than casting the reference manually, you can add the PreviousPageType directive to the .aspx page that receives the cross-page postback (in this example, CrossPage2.aspx), right after the Page directive. The PreviousPageType directive indicates the expected type of the page initiating the cross-page postback. Here's an example:

```
<%@ PreviousPageType VirtualPath="~/CrossPage1.aspx" %>
```

Now, the PreviousPage property will automatically use the CrossPage1 type. That allows you to skip the casting code and go straight to work using the previous page object, like this:

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

Notes

```
{
if (PreviousPage != null)
{
// (Read some information from the previous page.)
}
}
```

However, this approach is more fragile because it limits you to a single page class. You don't have the flexibility to deal with situations where more than one page might trigger a cross-page postback. For that reason, it's usually more flexible to use the casting approach.

Once you've cast the previous page to the appropriate page type, you still won't be able to directly access the control objects it contains. That's because the controls on the web page are not publicly accessible to other classes. You can work around this by using properties.

Example: If you want to expose the values from two text boxes in the source page, you might add properties that wrap the control variables. Here are two properties you could add to the CrossPage1 class to expose its TextBox controls:

```
public TextBox FirstNameTextBox
{
get { return txtFirstName; }
}
public TextBox LastNameTextBox
{
get { return txtLastName; }
}
```

However, this usually isn't the best approach. The problem is that it exposes too many details, giving the target page the freedom to read everything from the text in the text box to its fonts and colors. If you need to change the page later to use different input controls, it will be difficult to maintain these properties. Instead, you'll probably be forced to rewrite code in both pages.

A better choice is to define specific, limited methods or properties that extract just the information you need. For example, you might decide to add a FullName property that retrieves just the text from the two text boxes. Here's the full page code for CrossPage1.aspx with this property:

```
public partial class CrossPage1 : System.Web.UI.Page
{
public string FullName
{
get { return txtFirstName.Text + " " + txtLastName.Text; }
}
}
```

This way, the relationship between the two pages is clear, simple, and easy to maintain. You can probably change the controls in the source page (CrossPage1) without needing to change other parts of your application. For example, if you decide to use different controls for name entry in CrossPage1.aspx, you will be forced to revise the code for the FullName property. However, your changes would be confined to CrossPage1.aspx, and you wouldn't need to modify CrossPage2.aspx at all.

Notes

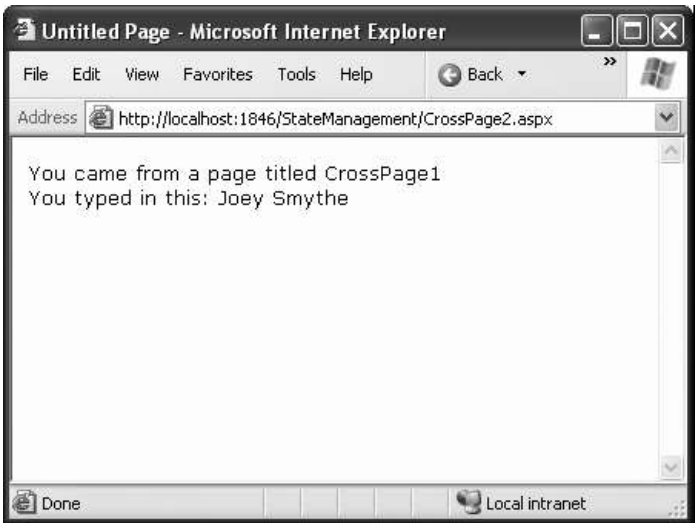
Here's how you can rewrite the code in CrossPage2.aspx to display the information from CrossPage1.aspx:

```
protected void Page_Load(object sender, EventArgs e)
{
if (PreviousPage != null)
{
lblInfo.Text = "You came from a page titled " +
PreviousPage.Title + "<br />";
CrossPage1 prevPage = PreviousPage as CrossPage1;
if (prevPage != null)
{
lblInfo.Text += "You typed in this: " + prevPage.FullName;
}
}
}
```

Notice that the target page (CrossPage2.aspx) can access the Title property of the previous page (CrossPage1.aspx) without performing any casting. That's because the Title property is defined as part of the base System.Web.UI.Page class, and so every web page includes it. However, to get access to the more specialized FullName property you need to cast the previous page to the right page class (CrossPage1), or use the PreviousPageType directive that was discussed earlier.

Figure 5.4 shows the new result.

Figure 5.4: Retrieving Specific Information from the Source Page



Notes Cross-page postbacks are genuinely useful, but they can lead the way to more complicated pages. If you allow multiple source pages to post to the same destination page, it's up to you to code the logic that figures out which page the user came from and then act accordingly. To avoid these headaches, it's easiest to perform cross-page postbacks between two specific pages only.

Notes

ASP.NET uses some interesting sleight of hand to make cross-page postbacks work. The first time the second page accesses Page.PreviousPage, ASP.NET needs to create the previous page object. To do this, it actually starts the page processing but interrupts it just before the PreRender stage, and it doesn't let the page render any HTML output.

However, this still has some interesting side effects. For example, all the page events of the previous page are fired, including Page.Load and Page.Init, and the Button.Click event also fires for the button that triggered the cross-page postback. ASP.NET fires these events because they might be needed to return the source page to the state it was last in, just before it triggered the cross-page postback.

Task

Create a web page which consists of multiple windows, each of which contains multiple frames. Place a command button on each frame that tells various information about the document contained in that frame including its title, URL, and so on.

5.2.2 The Query String

Another common approach is to pass information using a query string in the URL. This approach is commonly found in search engines. For example, if you perform a search on the Google website, you'll be redirected to a new URL that incorporates your search parameters. Here's an example:

```
http://www.google.ca/search?q=organic+gardening
```

The query string is the portion of the URL after the question mark. In this case, it defines a single variable named q, which contains the string organic+gardening.

The advantage of the query string is that it's lightweight and doesn't exert any kind of burden on the server. However, it also has several limitations:

1. Information is limited to simple strings, which must contain URL-legal characters.
2. Information is clearly visible to the user and to anyone else who cares to eavesdrop on the Internet.
3. The enterprising user might decide to modify the query string and supply new values, which your program won't expect and can't protect against.
4. Many browsers impose a limit on the length of a URL (usually from 1KB to 2KB). For that reason, you can't place a large amount of information in the query string and still be assured of compatibility with most browsers.

Adding information to the query string is still a useful technique. It's particularly well suited in database applications where you present the user with a list of items that correspond to records in a database, such as products. The user can then select an item and be forwarded to another page with detailed information about the selected item. One easy way to implement this design is to have the first page send the item ID to the second page. The second page then looks that item up in the database and displays the detailed information. You'll notice this technique in e-commerce sites such as Amazon.

To store information in the query string, you need to place it there yourself. Unfortunately, you have no collection-based way to do this. Typically, this means using a special HyperLink control or a special Response.Redirect() statement such as the one shown here:

```
// Go to newpage.aspx. Submit a single query string argument
```

Notes

```
// named recordID, and set to 10.
Response.Redirect("newpage.aspx?recordID=10");
```

You can send multiple parameters as long as they're separated with an ampersand (&):

```
// Go to newpage.aspx. Submit two query string arguments:
// recordID (10) and mode (full).
Response.Redirect("newpage.aspx?recordID=10&mode=full");
```

The receiving page has an easier time working with the query string. It can receive the values from the QueryString dictionary collection exposed by the built-in Request object:

```
string ID = Request.QueryString["recordID"];
```

Note that information is always retrieved as a string, which can then be converted to another simple data type. Values in the QueryString collection are indexed by the variable name. If you attempt to retrieve a value that isn't present in the query string, you'll get a null reference.

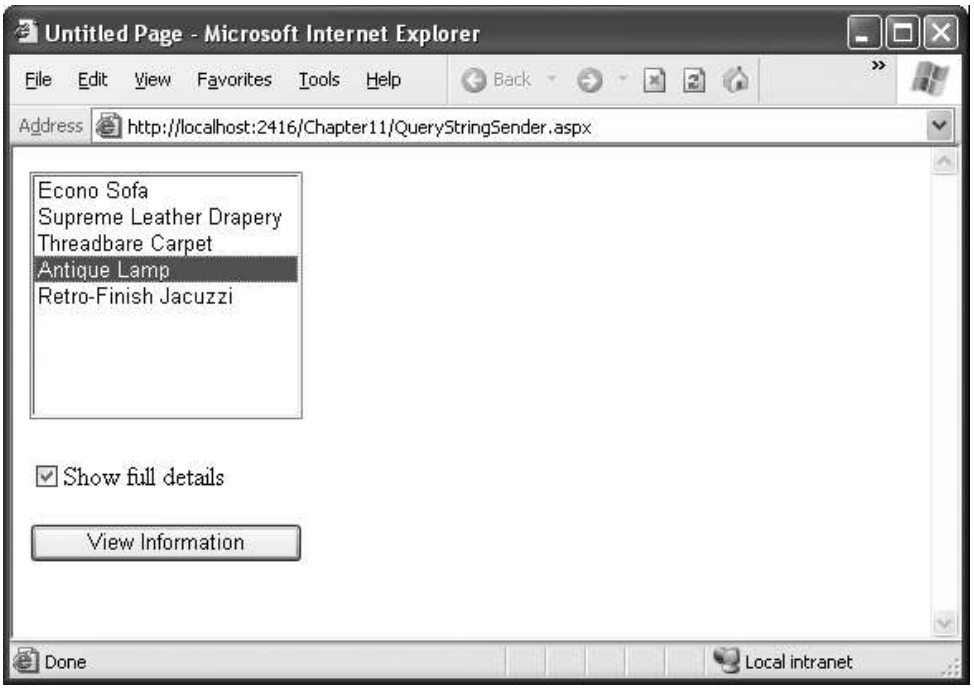
Notes Unlike view state, information passed through the query string is clearly visible and unencrypted. Don't use the query string for information that needs to be hidden or made tamperproof.

A Query String Example

When the user chooses an item by clicking the appropriate item in the list, the user is forwarded to a new page. This page displays the received ID number. This provides a quick and simple query string test with two pages.

The first page provides a list of items, a check box, and a submission button (Figure 5.5).

Figure 5.5: A Query String Sender



Notes

Here's the code for the first page:

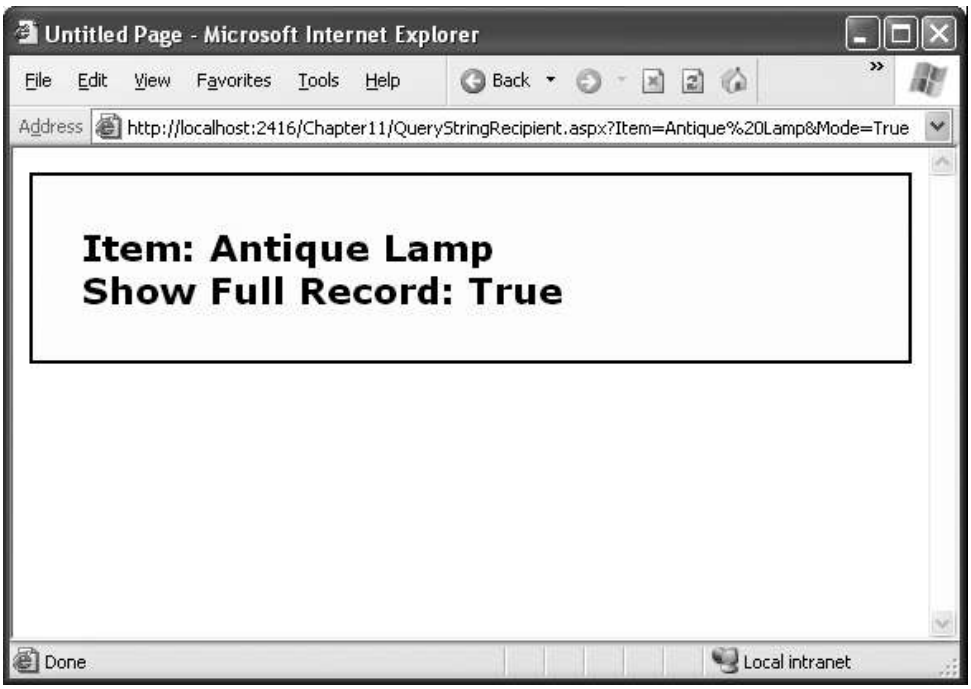
```
public partial class QueryStringSender : System.Web.UI.Page
{
protected void Page_Load(Object sender, EventArgs e)
{
if (!this.IsPostBack)
{
// Add sample values.
lstItems.Items.Add("Econo Sofa");
lstItems.Items.Add("Supreme Leather Drapery");
lstItems.Items.Add("Threadbare Carpet");
lstItems.Items.Add("Antique Lamp");
lstItems.Items.Add("Retro-Finish Jacuzzi");
}
}

protected void cmdGo_Click(Object sender, EventArgs e)
{
if (lstItems.SelectedIndex == -1)
{
lblError.Text = "You must select an item.";
}
else
{
// Forward the user to the information page,
// with the query string data.
string url = "QueryStringRecipient.aspx?";
url += "Item=" + lstItems.SelectedItem.Text + "&";
url += "Mode=" + chkDetails.Checked.ToString();
Response.Redirect(url);
}
}
}
```

Here's the code for the recipient page (Figure 5.6):

```
public partial class QueryStringRecipient : System.Web.UI.Page
{
protected void Page_Load(Object sender, EventArgs e)
{
lblInfo.Text = "Item: " + Request.QueryString["Item"];
lblInfo.Text += "<br />Show Full Record: ";
lblInfo.Text += Request.QueryString["Mode"];
}
}
```

Notes

Figure 5.6: A Query String Recipient



One interesting aspect of this example is that it places information in the query string that isn't valid namely, the space that appears in the item name. When you run the application, you'll notice that ASP.NET encodes the string for you automatically, converting spaces to the valid %20 equivalent escape sequence. The recipient page reads the original values from the QueryString collection without any trouble. This automatic encoding isn't always sufficient. To deal with special characters, you should use the URL-encoding technique.

URL Encoding

One potential problem with the query string is that some characters aren't allowed in a URL. In fact, the list of characters that are allowed in a URL is much shorter than the list of allowed characters in an HTML document. All characters must be alphanumeric or one of a small set of special characters (including $-_.+!*'(),). Some browsers tolerate certain additional special characters (Internet Explorer is notoriously lax), but many do not. Furthermore, some characters have special meaning. For example, the ampersand (&) is used to separate multiple query string parameters, the plus sign (+) is an alternate way to represent a space, and the number sign (#) is used to point to a specific bookmark in a web page. If you try to send query string values that include any of these characters, you'll lose some of your data. You can test this out with the previous example by adding items with special characters in the list box.

To avoid potential problems, it's a good idea to perform URL encoding on text values before you place them in the query string. With URL encoding, special characters are replaced by escaped character sequences starting with the percent sign (%), followed by a two-digit hexadecimal representation. For example, the & character becomes %26. The only exception is the space character, which can be represented as the character sequence %20 or the + sign.

To performURL encoding, you use the UrlEncode() and UrlDecode() methods of the HttpServerUtility class. As you learned in Unit 5, an HttpServerUtility object is made available to your code in every web form through the Page.Server property. The following code uses the

Notes

UrlEncode() method to rewrite the previous example, so it works with product names that contain special characters:

```
string url = "QueryStringRecipient.aspx?";
url += "Item=" + Server.UrlEncode(lstItems.SelectedItem.Text) + "&";
url += "Mode=" _ chkDetails.Checked.ToString();
Response.Redirect(url);
```

Notice that it's important not to encode everything. In this example, you can't encode the & character that joins the two query string values, because it truly is a special character.

You can use the UrlDecode() method to return a URL-encoded string to its initial value. However, you don't need to take this step with the query string. That's because ASP.NET automatically decodes your values when you access them through the Request.QueryString collection. (Many people still make the mistake of decoding the query string values a second time. Usually, decoding already decoded data won't cause a problem. The only exception is if you have a value that includes the + sign. In this case, using UrlDecode() will convert the + sign to a space, which isn't what you want.)

5.3 Cookies

Cookies provide another way that you can store information for later use. Cookies are small files that are created on the client's hard drive (or, if they're temporary, in the web browser's memory). One advantage of cookies is that they work transparently without the user being aware that information needs to be stored. They also can be easily used by any page in your application and even be retained between visits, which allows for truly long-term storage. They suffer from some of the same drawbacks that affect query strings—namely, they're limited to simple string information, and they're easily accessible and readable if the user finds and opens the corresponding file. These factors make them a poor choice for complex or private information or large amounts of data.

Some users disable cookies on their browsers, which will cause problems for web applications that require them. Also, users might manually delete the cookie files stored on their hard drives. But for the most part, cookies are widely adopted and used extensively on many websites.

Before you can use cookies, you should import the System.Net namespace so you can easily work with the appropriate types:

```
using System.Net;
```

Cookies are fairly easy to use. Both the Request and Response objects (which are provided through Page properties) provide a Cookies collection. The important trick to remember is that you retrieve cookies from the Request object, and you set cookies using the Response object.

To set a cookie, just create a new HttpCookie object. You can then fill it with string information (using the familiar dictionary pattern) and attach it to the current web response:

```
// Create the cookie object.
HttpCookie cookie = new HttpCookie("Preferences");

// Set a value in it.
cookie["LanguagePref"] = "English";

// Add another value.
cookie["Country"] = "US";
```

Notes

```
// Add it to the current web response.
Response.Cookies.Add(cookie);
```

A cookie added in this way will persist until the user closes the browser and will be sent with every request. To create a longer-lived cookie, you can set an expiration date:

```
// This cookie lives for one year.
cookie.Expires = DateTime.Now.AddYears(1);
```

You retrieve cookies by cookie name using the Request.Cookies collection:

```
HttpCookie cookie = Request.Cookies["Preferences"];
// Check to see whether a cookie was found with this name.
// This is a good precaution to take,
// because the user could disable cookies,
// in which case the cookie will not exist.
string language;
if (cookie != null)
{
language = cookie["LanguagePref"];
}
```

The only way to remove a cookie is by replacing it with a cookie that has an expiration date that has already passed. This code demonstrates the technique:

```
HttpCookie cookie = new HttpCookie("LanguagePref");
cookie.Expires = DateTime.Now.AddDays(-1);
Response.Cookies.Add(cookie);
```

A Cookie Example

The next example shows a typical use of cookies to store a customer name. If the name is found, a welcome message is displayed, as shown in Figure 5.7.

Figure 5.7: Displaying Information from a Custom Cookie

Notes

Here's the code for this page:

```
public partial class CookieExample : System.Web.UI.Page
{
protected void Page_Load(Object sender, EventArgs e)
{
HttpCookie cookie = Request.Cookies["Preferences"];
if (cookie == null)
{
lblWelcome.Text = "<b>Unknown Customer</b>";
}
else
{
lblWelcome.Text = "<b>Cookie Found.</b><br /><br />";
lblWelcome.Text += "Welcome, " + cookie["Name"];
}
}
protected void cmdStore_Click(Object sender, EventArgs e)
{
// Check for a cookie, and only create a new one if
// one doesn't already exist.
HttpCookie cookie = Request.Cookies["Preferences"];
if (cookie == null)
{
cookie = new HttpCookie("Preferences");
}
cookie["Name"] = txtName.Text;
cookie.Expires = DateTime.Now.AddYears(1);
Response.Cookies.Add(cookie);
lblWelcome.Text = "<b>Cookie Created.</b><br /><br />";
lblWelcome.Text += "New Customer: " + cookie["Name"];
}
}
```

Notes You'll find that some other ASP.NET features use cookies. Two examples are session state (which allows you to temporarily store user-specific information in server memory) and forms security (which allows you to restrict portions of a website and force users to access it through a login page).

5.4 Session State

There comes a point in the life of most applications when they begin to have more sophisticated storage requirements. An application might need to store and access complex information such as custom data objects, which can't be easily persisted to a cookie or sent through a query string. Or the application might have stringent security requirements that prevent it from storing

Notes

information about a client in view state or in a custom cookie. In these situations, you can use ASP.NET's built-in session state facility.

Session state management is one of ASP.NET's premiere features. It allows you to store any type of data in memory on the server. The information is protected, because it is never transmitted to the client, and it's uniquely bound to a specific session. Every client that accesses the application has a different session and a distinct collection of information. Session state is ideal for storing information such as the items in the current user's shopping basket when the user browses from one page to another.

5.4.1 Session Tracking

ASP.NET tracks each session using a unique 120-bit identifier. ASP.NET uses a proprietary algorithm to generate this value, thereby guaranteeing (statistically speaking) that the number is unique and it's random enough that a malicious user can't reverse-engineer or "guess" what session ID a given client will be using. This ID is the only piece of session-related information that is transmitted between the web server and the client.

When the client presents the session ID, ASP.NET looks up the corresponding session, retrieves the objects you stored previously, and places them into a special collection so they can be accessed in your code. This process takes place automatically.

For this system to work, the client must present the appropriate session ID with each request. You can accomplish this in two ways:

1. ***Using cookies***: In this case, the session ID is transmitted in a special cookie (named ASP.NET_SessionId), which ASP.NET creates automatically when the session collection is used. This is the default, and it's also the same approach that was used in earlier versions of ASP.
2. ***Using modified URLs***: In this case, the session ID is transmitted in a specially modified (or munged) URL. This allows you to create applications that use session state with clients that don't support cookies.

Session state doesn't come for free. Though it solves many of the problems associated with other forms of state management, it forces the server to store additional information in memory. This extra memory requirement, even if it is small, can quickly grow to performance destroying levels as hundreds or thousands of clients access the site.

In other words, you must think through any use of session state. A careless use of session state is one of the most common reasons that a web application can't scale to serve a large number of clients.

5.4.2 Using Session State

You can interact with session state using the System.Web.SessionState.HttpSessionState class, which is provided in an ASP.NET web page as the built-in Session object. The syntax for adding items to the collection and retrieving them is basically the same as for adding items to a page's view state.

Example: You might store a DataSet in session memory like this:

```
Session["InfoDataSet"] = dsInfo;
```

You can then retrieve it with an appropriate conversion operation:

```
dsInfo = (DataSet)Session["InfoDataSet"];
```

Notes

Session state is global to your entire application for the current user. However, session state can be lost in several ways:

1. If the user closes and restarts the browser.
2. If the user accesses the same page through a different browser window, although the session will still exist if a web page is accessed through the original browser window. Browsers differ on how they handle this situation.
3. If the session times out due to inactivity. More information about session timeout can be found in the configuration section.
4. If your web page code ends the session by calling the Session.Abandon() method.

In the first two cases, the session actually remains in memory on the web server, because ASP.NET has no idea that the client has closed the browser or changed windows. The session will linger in memory, remaining inaccessible, until it eventually expires.

Table 5.1 describes the methods and properties of the HttpSessionState class.

Table 5.1: HttpSessionState Members

Property	Description
Count	Provides the number of items in the current session collection.
IsCookieless	Identifies whether the session is tracked with a cookie or modified URLs.
IsNewSession	Identifies whether the session was created only for the current request. If no information is in session state, ASP.NET won't bother to track the session or create a session cookie. Instead, the session will be re-created with every request.
Mode	Provides an enumerated value that explains how ASP.NET stores session state information. This storage mode is determined based on the web.config settings.
SessionID	Provides a string with the unique session identifier for the current client.
Timeout	Determines the number of minutes that will elapse before the current session is abandoned, provided that no more requests are received from the client. This value can be changed programmatically, letting you make the session collection longer when needed.
Abandon()	Cancels the current session immediately and releases all the memory it occupied. This is a useful technique in a logoff page to ensure that server memory is reclaimed as quickly as possible.
Clear()	Removes all the session items but doesn't change the current session identifier.

5.5 Session State Configuration

You configure session state through the web.config file for your current application (which is found in the same virtual directory as the .aspx web page files). The configuration file allows you to set advanced options such as the timeout and the session state mode. If you're creating your web application in Visual Studio, your project will include an automatically generated web.config file.

The following listing shows the most important options that you can set for the <sessionState> element. Keep in mind that you won't use all of these details at the same time. Some settings only apply to certain session state modes, as you'll see shortly.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
```

Notes

```
<system.web>
<!-- Other settings omitted. -->

<sessionState
cookieless="UseCookies" cookieName="ASP.NET_SessionId"
regenerateExpiredSessionId="false"
timeout="20"
mode="InProc"
stateConnectionString="tcpip=127.0.0.1:42424"
stateNetworkTimeout="10"
sqlConnectionString="data source=127.0.0.1;Integrated Security=SSPI"
sqlCommandTimeout="30"
allowCustomSqlDatabase="false"
customProvider=""
/>
</system.web>
</configuration>
```

The following sections describe the preceding session state settings.

Task

Use a timer function to add items to a menu from VBScript. Be sure to check for the menu object's existence with ISObject before adding items to the menu.

5.5.1 Cookieless

You can set the cookieless setting to one of the values defined by the HttpCookieMode enumeration, as described in Table 5.2.

Table 5.2: HttpCookieMode Values

Value	Description
UseCookies	Cookies are always used, even if the browser or device doesn't support cookies or they are disabled. This is the default. If the device does not support cookies, session information will be lost over subsequent requests, because each request will get a new ID.
UseUri	Cookies are never used, regardless of the capabilities of the browser or device. Instead, the session ID is stored in the URL.
UseDeviceProfile	ASP.NET chooses whether to use cookieless sessions by examining the BrowserCapabilities object. The drawback is that this object indicates what the device should support—it doesn't take into account that the user may have disabled cookies in a browser that supports them.
AutoDetect	ASP.NET attempts to determine whether the browser supports cookies by attempting to set and retrieve a cookie (a technique commonly used on the Web). This technique can correctly determine whether a browser supports cookies but has them disabled, in which case cookieless mode is used instead.

Here's an example that forces cookieless mode (which is useful for testing):

```
<sessionState cookieless="UseUri" ... />
```

Notes

In cookieless mode, the session ID will automatically be inserted into the URL. When ASP.NET receives a request, it will remove the ID, retrieve the session collection, and forward the request to the appropriate directory. Figure 5.8 shows a munged URL.

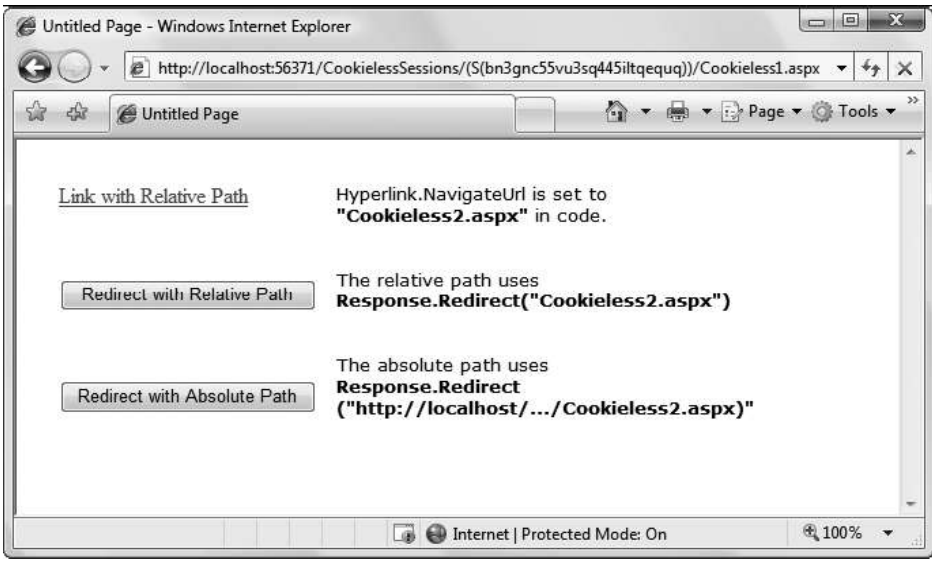
Figure 5.8: A Munged URL with the Session ID

Because the session ID is inserted in the current URL, relative links also automatically gain the session ID. In other words, if the user is currently stationed on Page1.aspx and clicks a relative link to Page2.aspx, the relative link includes the current session ID as part of the URL. The same is true if you call Response.Redirect() with a relative URL, as shown here:

```
Response.Redirect("Page2.aspx");
```

Figure 5.9 shows a sample website (included with the online samples in the CookielessSessions directory) that tests cookieless sessions. It contains two pages and uses cookieless mode. The first page (Cookieless1.aspx) contains a HyperLink control and two buttons, all of which take you to a second page (Cookieless2.aspx). The trick is that these controls have different ways of performing their navigation. Only two of them work with cookieless session the third loses the current session.

Figure 5.9: Three Tests of Cookieless Sessions



The HyperLink control navigates to the page specified in its NavigateUrl property, which is set to the relative path Cookieless2.aspx. If you click this link, the session ID is retained in the URL, and the new page can retrieve the session information. This demonstrates that cookieless sessions work with relative links.

The two buttons on this page use programmatic redirection by calling the Response.Redirect() method. The first button uses the relative path Cookieless2.aspx, much like the HyperLink control. This approach works with cookieless session state, and preserves the munged URL with no extra steps required.

Notes

```
protected void cmdLink_Click(Object sender, EventArgs e)
{
Response.Redirect("Cookieless2.aspx");
}
```

The only real limitation of cookieless state is that you cannot use absolute links (links that include the full URL, starting with http://). The second button uses an absolute link to demonstrate this problem. Because ASP.NET cannot insert the session ID into the URL, the session is lost.

```
protected void cmdLinkAbsolute_Click(Object sender, EventArgs e)
{
string url = "http://localhost:56371/CookielessSessions/Cookieless2.aspx";
Response.Redirect(url);
}
```

Now the target page (Figure 5.10) checks for the session information, but can't find it.

Figure 5.10: A Lost Session

Writing the code to demonstrate this problem in a test environment is a bit tricky. The problem is that Visual Studio's integrated web server chooses a different port for your website every time you start it. As a result, you'll need to edit the code every time you open Visual Studio so that your URL uses the right port number (such as 56371 in the previous example).

There's another workaround. You can use some crafty code that gets the current URL from the page and just modifies the last part of it (changing the page name from Cookieless1.aspx

```
// Create a new URL based on the current URL (but ending with
// the page Cookieless2.aspx instead of Cookieless1.aspx.
string url = "http://" + Request.Url.Authority +
Request.Url.Segments[0] + Request.Url.Segments[1] +
"Cookieless2.aspx";
Response.Redirect(url);
```

Notes

Of course, if you deploy your website to a real virtual directory that's hosted by IIS, you won't use a randomly chosen port number anymore, and you won't experience this quirk.

5.5.2 Timeout

Another important session state setting in the web.config file is the timeout. This specifies the number of minutes that ASP.NET will wait, without receiving a request, before it abandons the session.

This setting represents one of the most important compromises of session state. A difference of minutes can have a dramatic effect on the load of your server and the performance of your application. Ideally, you will choose a timeframe that is short enough to allow the server to reclaim valuable memory after a client stops using the application but long enough to allow a client to pause and continue a session without losing it.

You can also programmatically change the session timeout in code. For example, if you know a session contains an unusually large amount of information, you may need to limit the amount of time the session can be stored. You would then warn the user and change the timeout property. Here's a sample line of code that changes the timeout to 10 minutes:

```
Session.Timeout = 10;
```

5.5.3 Mode

The remaining session state settings allow you to configure ASP.NET to use different session state services, depending on the mode that you choose. The next few sections describe the modes you can choose from.

Notes If you're hosting ASP.NET using more than one web server (which is affectionately known as a web farm), you'll also need to take some extra configuration steps to make sure all the web servers are in sync. Otherwise, one server might encode information in session state differently than another, which will cause a problem if the user is routed from one server to another during a session. The solution is to modify the <machineKey> section of the machine.config file so that it's consistent across all servers.

InProc

InProc is the default mode. InProc is similar to how session state was stored in previous versions of ASP. It instructs information to be stored in the same process as the ASP.NET worker threads, which provides the best performance but the least durability. If you restart your server, the state information will be lost.

InProc makes sense for most small websites. In a web farm scenario, though, it won't work. To allow session state to be shared between servers, you must use the out-of-process or SQL Server state service. Another reason you might want to avoid InProc mode is if you find that your users are losing session state information at unpredictable times. In ASP.NET, application domains can be restarted for a variety of reasons, including configuration changes and updated pages, and when certain thresholds are met (regardless of whether an error has occurred). If you find that you're losing sessions before the timeout limit, you may want to experiment with a more durable mode.

Notes

Notes When using the StateServer and SQLServer modes, the objects you store in session state must be serializable. Otherwise, ASP.NET will not be able to transmit the object to the state service or store it in the database. Earlier in this unit , you learned how to create a serializable Customer class for storing in view state.

Off

This setting disables session state management for every page in the application. This can provide a slight performance improvement for websites that are not using session state.

StateServer

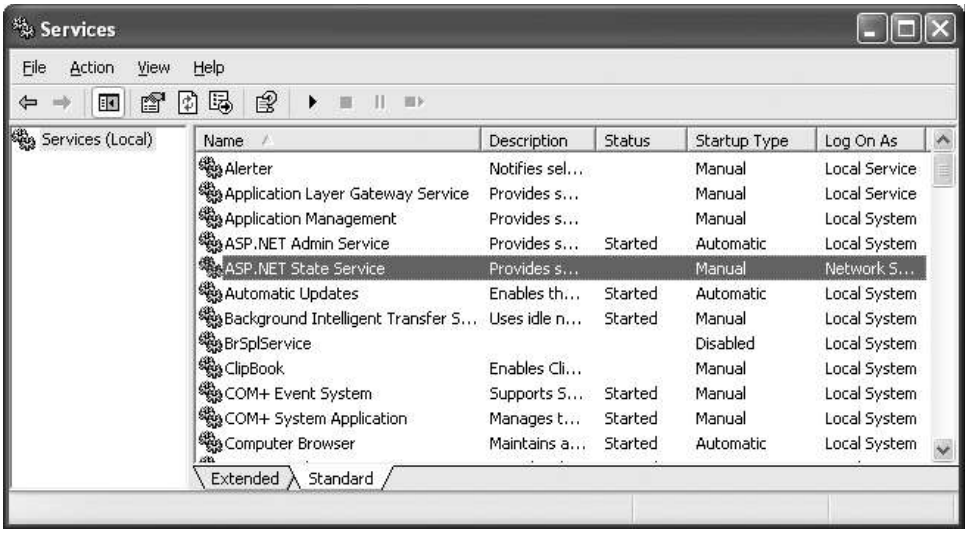
With this setting, ASP.NET will use a separate Windows service for state management. This service runs on the same web server, but it's outside the main ASP.NET process, which gives it a basic level of protection if the ASP.NET process needs to be restarted. The cost is the increased time delay imposed when state information is transferred between two processes. If you frequently access and change state information, this can make for a fairly unwelcome slowdown.

When using the StateServer setting, you need to specify a value for the stateConnectionString setting. This string identifies the TCP/IP address of the computer that is running the StateServer service and its port number (which is defined by ASP.NET and doesn't usually need to be changed). This allows you to host the StateServer on another computer. If you don't change this setting, the local server will be used (set as address 127.0.0.1).

Of course, before your application can use the service, you need to start it. The easiest way to do this is to use the Microsoft Management Console (MMC). Here's how:

1. Select Start → Settings → Control Panel (or just Start → Control Panel in Windows Vista).
2. Open the Administrative Tools group, and then choose Computer Management.
3. In the Computer Management tool, go to the Services and Applications → Services node.
4. Find the service called ASP.NET State Service in the list, as shown in Figure 5.11.

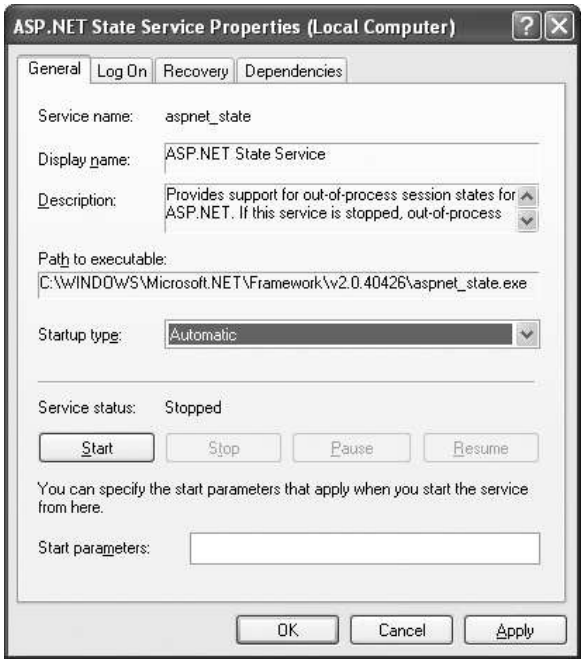
Figure 5.11: The ASP.NET State Service



Notes

5. Once you find the service in the list, you can manually start and stop it by right-clicking it. Generally, you'll want to configure Windows to automatically start the service. Rightclick it, select Properties, and modify the Startup Type, setting it to Automatic, as shown in Figure 5.12.

Figure 5.12: Windows Services



Notes When using StateServer mode, you can also set an optional stateNetworkTimeout attribute that specifies the maximum number of seconds to wait for the service to respond before canceling the request. he default value is 10 (seconds).

SQLServer

This setting instructs ASP.NET to use an SQL Server database to store session information, as identified by the sqlConnectionString attribute. This is the most resilient state store but also the slowest by far. To use this method of state management, you'll need to have a server with SQL Server installed.

When setting the sqlConnectionString attribute, you follow the same sort of pattern you use with ADO.NET data access. Generally, you'll need to specify a data source (the server address) and a user ID and password, unless you're using SQL integrated security.

In addition, you need to install the special stored procedures and temporary session databases. These stored procedures take care of storing and retrieving the session information. ASP.NET includes a command-line tool that does the work for you automatically, called aspnet_regsql.exe. It's found in the c:\Windows\Microsoft.NET\Framework\v2.0.50727 directory. The easiest way to run aspnet_regsql.exe is to start by launching the Visual Studio command prompt (open the Start menu and choose Programs →Visual Studio 2008 → Visual Studio Tools →Visual Studio 2008 Command Prompt). You can then type in an aspnet_regsql.exe command, no matter what directory you're in.

Notes

You can use the aspnet_regsql.exe tool to perform several database-related tasks. To use aspnet_regsql.exe to create a session storage database, you supply the –ssadd parameter. In addition, you use the –S parameter to indicate the database server name, and the –E parameter to log in to the database using the currently logged-in Windows user account.

Here's a command that creates the session storage database on the current computer, using the default database name ASPState:

```
aspnet_regsql.exe -S localhost -E -ssadd
```

This command uses the alias localhost, which tells aspnet_regsql.exe to connect to the database server on the current computer.

Notes The aspnet_regsql.exe command supports additional options that allow you to store session information in a database with a different name. You can find out about these options by referring to the Visual Studio help (look up aspnet_regsql in the index) or by surfing to http://msdn2.microsoft.com/en-us/ library/ms178586.aspx.

Once you've created your session state database, you need to tell ASP.NET to use it by modifying the <sessionState> section of the web.config file. If you're using a database named ASPState to store your session information (which is the default), you don't need to supply the database name. Instead, you simply have to indicate the location of the server and the type of authentication that ASP.NET should use to connect to it, as shown here:

```
<sessionState
sqlConnectionString="data source=127.0.0.1;Integrated Security=SSPI"
... />
```

When using the SQLServer mode, you can also set an optional sqlCommandTimeout attribute that specifies the maximum number of seconds to wait for the database to respond before canceling the request. The default is 30 seconds.

Custom

When using custom mode, you need to indicate which session state store provider to use by supplying the customProvider attribute. The customProvider attribute indicates the name of the class. The class may be part of your web application (in which case the source code is placed in the App_Code subfolder) or it can be in an assembly that your web application is sing (in which case the compiled assembly is placed in the Bin subfolder).

Creating a custom state provider is a low-level task that needs to be handled carefully to ensure security, stability, and scalability. Custom state providers are also beyond the scope of this book. However, other vendors may release custom state providers you want to use. For example, Oracle could provide a custom state provider that allows you to store state information in an Oracle database.

5.6 Application State

Application state allows you to store global objects that can be accessed by any client. Application state is based on the System.Web.HttpApplicationState class, which is provided in all web pages through the built-in Application object.

Application state is similar to session state. It supports the same type of objects, retains information on the server, and uses the same dictionary-based syntax. A common example with application

Notes

state is a global counter that tracks how many times an operation has been performed by all the web application's clients.

Example: You could create a Global.asax event handler that tracks how many sessions have been created or how many requests have been received into the application. Or you can use similar logic in the Page.Load event handler to track how many times a given page has been requested by various clients. Here's an example of the latter:

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

// Retrieve the current counter value.
int count = 0;
if (Application["HitCounterForOrderPage"] != null)
{
count = (int)Application["HitCounterForOrderPage"];
}

// Increment the counter.
count++;

// Store the current counter value.
Application["HitCounterForOrderPage"] = count;
lblCounter.Text = count.ToString();
}
```

Once again, application state items are stored as objects, so you need to cast them when you retrieve them from the collection. Items in application state never time out. They last until the application or server is restarted, or the application domain refreshes itself (because of automatic process recycling settings or an update to one of the pages or components in the application).

Application state isn't often used, because it's generally inefficient. In the previous example, the counter would probably not keep an accurate count, particularly in times of heavy traffic. For example, if two clients requested the page at the same time, you could have a sequence of events like this:

1. User A retrieves the current count (432).
2. User B retrieves the current count (432).
3. User A sets the current count to 433.
4. User B sets the current count to 433.

In other words, one request isn't counted because two clients access the counter at the same time. To prevent this problem, you need to use the Lock() and Unlock() methods, which explicitly allow only one client to access the Application state collection at a time.

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

// Acquire exclusive access.
Application.Lock();
```

Notes

```
int count = 0;
if (Application["HitCounterForOrderPage"] != null)
{
count = (int)Application["HitCounterForOrderPage"];
}
count++;
Application["HitCounterForOrderPage"] = count;

// Release exclusive access.
Application.Unlock();
lblCounter.Text = count.ToString();
}
```

Unfortunately, all other clients requesting the page will be stalled until the Application collection is released. This can drastically reduce performance. Generally, frequently modified values are poor candidates for application state. In fact, application state is rarely used in the .NET world because its two most common uses have been replaced by easier, more efficient methods:

1. In the past, application state was used to store application-wide constants, such as a database connection string.
2. Application state can also be used to store frequently used information that is timeconsuming to create, such as a full product catalog that requires a database lookup. However, using application state to store this kind of information raises all sorts of problems about how to check whether the data is valid and how to replace it when needed. It can also hamper performance if the product catalog is too large.

Case Study

The two developers wanted to create their new solution using a Model-View-Controller (MVC) approach, which simplifies application programming. MVC architecture, first described by Trygve Reekskaug in 1979, uses a design pattern based upon a model which provides a data representation relevant to the needs of the application; a view, or interface, for interacting with the data model; and a controller for mediating between the view and the model objects. The three together are sometimes referred to as a triplet, and MVC-based applications can be assembled as a series of triplets, each independent of the other.

The two wanted to create their site using the Microsoft Application Platform, but needed an MVC solution. "We were definitely sticking with the Microsoft stack because we like the Microsoft stack," Atwood says. "We're big fans of the language C#. Historically C# has evolved rapidly, and in ways that we like. To stay within C# we were preparing to create our own MVC implementation, but then we heard about the beta version of the Microsoft ASP.NET MVC Framework."

Solution:

Atwood and Spolsky created their new site—Stack Overflow—using the ASP.NET MVC Framework, the Visual Studio 2008 development system, and the Microsoft .NET Framework 3.5, which are all part of the Microsoft Web Platform. The site, which was

Contd...

Notes

created within two months, proved popular and was able to scale to support what turned out to be rapid growth.

"Stack Overflow has grown incredibly fast," Spolsky says. "After a year in business, it gets over a million page views most weekdays and currently stands as the 817th largest site on the Internet, according to Quantcast. It reaches 5.2 million people a month." At last count the site hosted 500,000 questions along with the related answers and discussions, ranked according to what the community felt were the best responses.

The two have repurposed the same code to broaden the scope beyond programmers, launching sister sites including Server Fault, for system administrators, and Super User, for computer "power users."

A spin off of the software, called StackExchange, has been released by Fog Creek Software to allow third parties to build Stack-Overflow communities around other topics. "You can use the same software to create a Q&A site about anything," Atwood says. "Someone could create a Stack Overflow Q&A community for airplanes, or stamp collecting, geology, or any other area of interest."

Stack Overflow was created using the Microsoft Web Platform, which in addition to development tools includes the Windows Server 2008 operating system, Internet Information Services (IIS) 7, and Microsoft SQL Server 2008 database software.

The basic elements of Stack Overflow include:

1. ***Load Balancing:*** Users entering the stackoverflow.com site are first handled by a load balancer, which assigns a user to one of several Web servers according to IP address. Users continue to be assigned to the same server for as long as they have the same IP address. Stack Overflow uses the open-source HAProxy for this load balancing.
2. ***Presentation:*** Customers browse pages served by Windows Server 2008 Standard, through IIS 7.5. The Web servers are computers with Xeon quad core CPUs and 8 gigabytes (GB) RAM. When a user submits a question or an answer, their information is written to the database using the lightweight LINQ to SQL database abstraction layer.
3. ***Database:*** The database is hosted on a dedicated server with 2 Intel Xeon quad core CPUs and 48 GB of RAM, and runs Microsoft SQL Server 2008 Enterprise. Stack Overflow takes advantage of the Online Indexing feature, introduced with SQL Server 2008 Enterprise, which enables table re-indexing while the database continues to run.

"We designed this to be a next generation Q&A site," Atwood says. "We tried to combine aspects of Wikis, in that everybody can edit everything once the system learns to trust them. Voting, as with DIGG and REDDIT, are an important component. Stack Overflow has a reputation system which comes out of the voting, so as other people vote your content up that increases your reputation score and it unlocks different abilities on the site as the system learns to trust you. It has elements of blogging in that there's a strong sense of ownership. When you create a question you're the owner of that question and you have special privileges as you curate that question. And, like a blog, people can post comments. We wanted to synthesize the best of what we saw in all of these types of communities and bring them all together into one simple site."

Contd...

Notes

Benefits

Building the Stack Overflow solution using the ASP.NET MVC Framework, the .NET Framework 3.5, Visual Studio 2008, and other elements of the Microsoft Web Platform gave the site the high performance and fast development that the developers had anticipated they would gain from using an MVC approach. Developers found that testing and code maintenance was easier using the ASP.NET MVC Framework. The company has also benefitted from the strong developer community that has emerged to support the ASP.NET MVC Framework.

High Performance

The Stack Overflow team was impressed by the performance of the site they created using the Microsoft Web Platform. "Our site is definitely faster than if we had created it with a more traditional abstraction-based Web architecture," Atwood says. "The .NET Framework is responsible for much of the speed we see, but ASP.NET MVC plays a role because it is so lightweight. It enhances performance."

The high performance also contributes to the real-world scalability Stack Overflow has enjoyed. "I want to emphasize that MVC not only works from an architectural standpoint of clean code and sensibility, but it also scales really well in the wild," Atwood says.

Fast Development

Stack Overflow developers credit the clean separation between presentation code, business logic, and the data model they gained by using the ASP.NET MVC Framework with enabling faster development than they could have achieved using abstraction-based Web development models. The developers also praised the low-level HTTP control they gained using ASP.NET MVC.

"The first thing that saved us time was that Microsoft created an MVC framework so that we didn't have to create our own," Atwood says. "ASP.NET MVC provides a simple framework that enables developers to work very efficiently. It mapped very well to our mental model of what Web development should be."

The developers liked the design of ASP.NET MVC. "Microsoft's MVC implementation is well designed in the sense that it anticipates all the standard work a Web developer would have to do," Atwood says. "So there is great coverage for the basics, and a clean framework that enables you to go far beyond the basics. We never felt that there was something that we couldn't do within the MVC framework."

The openness of the framework was appreciated by the developers. "ASP.NET MVC is very pluggable," Atwood says. "Microsoft allows you to use your own viewer, for example. One of our developers loves jQuery [an open-source JavaScript library] and so we use jQuery, which is supported by Microsoft's implementation of MVC. We also like that Microsoft has posted the source code for ASP.NET MVC on the CodePlex open-source site."

Easier Testing and Code Maintenance

The Stack Overflow team is enjoying the separation of concerns that is inherent to an MVC approach to development, finding that code written with an MVC pattern is easier to test and maintain because logic isn't integrated into the view state.

"With MVC you have a more logical structure for the code," Atwood says. "By keeping application logic in the controller or data model, it is easier to create, test, and maintain the code. The more traditional method of embedding a lot of code into the view makes for

Contd...

Notes

cumbersome development and can create difficult testing scenarios. MVC enables a more direct control of HTML than working through an abstraction layer, and this also makes it easier to test and maintain the code."

Strong Community

Atwood says he enjoys the strong community he sees developing around the ASP.NET MVC Framework. Developers are excited to have an MVC implementation designed to interoperate with Visual Studio, the .NET Framework, and other elements of the Microsoft Web Platform. He identified Microsoft's posting of its MVC source code to CodePlex as representing a large contribution to the sense of community that is developing around the product.

"During the beta phase, some common bugs were identified by members of the developer community," Atwood says. "The difference was that this time we didn't have to wait for Microsoft to post the bug fixes. Someone in the community just posted their own fix, because they had access to the code. The community is way faster than a company can be. We just downloaded the fix and were on our merry way."

With Microsoft at the core of the community, Atwood likes the fact that Microsoft will continue to enhance the MVC Framework.

"A basic rule of programming is to never write what you can buy or get for free," Atwood says. "An MVC implementation was needed. Microsoft created one, and it is generally accepted within the Microsoft developer community that they did a great job on this. It is reassuring to know that Microsoft will continue to build upon and support its MVC framework because we're huge fans of the .NET Framework, C#, and Visual Studio. We love these tools. Part of the reason we built Stack Overflow was to show people: 'Look, you can build really awesome stuff with this stack.'"

5.7 Summary

- State management is the art of retaining information between requests.
- Usually, this information is user-specific (such as a list of items in a shopping cart, a user name, or an access level), but sometimes it's global to the whole application (such as usage statistics that track site activity).
- Because ASP.NET uses a disconnected architecture, you need to explicitly store and retrieve state information with each request.
- The approach you choose to store this data can dramatically affect the performance, scalability, and security of your application.

5.8 Keywords

Application Base: The directory where a .NET application's assembly files are stored.

Application State: In ASP.NET, a variable store that is created on the server for the current application and is shared by all users. Application state is typically used to store information that is used for all users, such as application-wide settings.

Class: In .NET languages, classes are templates used for defining new types. Classes describe both the properties and behaviors of objects. Properties contain the data that are exposed by the class.

Notes

Event: A notification by a program or operating system that "something has happened." An event may be fired (or raised) in response to the occurrence of a pre-defined action (e.g., a window getting focus, a user clicking a button, a timer indicating a specific interval of time has passed, or a program starting up or shutting down).

Session State: In ASP.NET, a variable store created on the server for the current user; each user maintains a separate Session state on the server. Session state is typically used to store user-specific information between postbacks.

5.9 Self Assessment

Fill in the blanks:

1. ..... uses a hidden field that ASP.NET automatically inserts in the final, rendered HTML of a web page.
2. The ViewState property of the page provides the .....
3. A ..... postback is a technique that extends the postback mechanism.
4. ..... approach is commonly found in search engines.
5. The ..... has an easier time working with the query string.
6. ..... are small files that are created on the client's hard drive.
7. Both the ..... and Response objects provide a Cookies collection.

State whether the following statements are True or False:

8. Session state management is one of ASP.NET's premiere features.
9. ASP.NET do not tracks each session using a unique 120-bit identifier.
10. The configuration file allows you to set advanced options such as the timeout and the session state mode.

5.10 Review Questions

1. What is a session?
2. What is the state of a page?
3. What setting can you use to see information about the different stages of the page life cycle?
4. In which life cycle phase does the page determine if it was called as the result of a postback?
5. What event is usually used to take actions during the Load phase?
6. What are the four kinds of state that ASP.NET manages? Which one can you not affect in any way?
7. What does the EnableViewState="false" setting do?
8. Where would you store the value of a counter that is incremented each time the page is loaded?
9. Suppose you wanted to ask the user to enter her name on a page, and you wanted to retain that value the entire time the user is at your site. What's the best mechanism to use?
10. What's the proper syntax for storing and retrieving that user name?

Notes

Answers: Self Assessment

1.	View state	2.	current view state information
3.	cross-page	4.	Query string
5.	receiving page	6.	Cookies
7.	Request	8.	True
9.	False	10.	True

5.11 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Notes

Unit 6: Error Handling Logging and Tracing

CONTENTS

Objectives

Introduction

6.1 Common Errors

6.2 Exception Handling

6.2.1 The Exception Class

6.2.2 The Exception Chain

6.3 Handling Exceptions

6.3.1 Catching Specific Exceptions

6.3.2 Nested Exception Handlers

6.3.3 Exception Handling in Action

6.4 Throwing your own Exceptions

6.5 Logging Exceptions

6.5.1 Viewing the Windows Event Logs

6.5.2 Writing to the Event Log

6.5.3 Custom Logs

6.5.4 A Custom Logging Class

6.5.5 Retrieving Log Information

6.6 Error Pages

6.6.1 Error Modes

6.6.2 Custom Error Pages

6.7 Page Tracing

6.7.1 Enabling Tracing

6.7.2 Tracing Information

6.8 Summary

6.9 Keywords

6.10 Self Assessment

6.11 Review Questions

6.12 Further Readings

Notes

Objectives

After studying this unit, you will be able to:

- Define common errors
- Describe handling exceptions
- Explain throwing your own exceptions
- Know logging exceptions
- Define page tracing

Introduction

Before ASP 3.0, error handling was never a strong suit of ASP. Despite taking great efforts to handle possible error conditions, it is not uncommon to see ASP applications crash and display cryptic error messages. For applications critical to a company's success, this is a huge embarrassment. You may have seen something like this quite often:

```
Microsoft VBScript runtime error '800a0006'
Overflow
/wad/vote.asp, line 25
```

Besides handling errors, how many times have you forgotten to remove debugging statements in your application? Often, due to the unrealistic and tight deadlines imposed by management, you end up rushing to deploy your application. In the midst of doing that, a few debugging statements occasionally get left out.

In this unit, you'll learn the error-handling and debugging practices that you can use to defend your ASP.NET applications against common errors, track user problems, and solve mysterious issues. You'll learn how to use structured exception handling, how to use logs to keep a record of unrecoverable errors, and how to set up web pages with custom error messages for common HTTP errors.

6.1 Common Errors

Errors can occur in a variety of situations. Some of the most common causes of errors include attempts to divide by zero (usually caused by invalid input or missing information) and attempts to connect to a limited resource such as a file or a database (which can fail if the file doesn't exist, the database connection times out, or the code has insufficient security credentials).

One infamous type of error is the null reference exception, which usually occurs when a program attempts to use an uninitialized object. As a .NET programmer, you'll quickly learn to recognize and resolve this common but annoying mistake. The following code example shows the problem in action, with two SqlConnection objects that represent database connections:

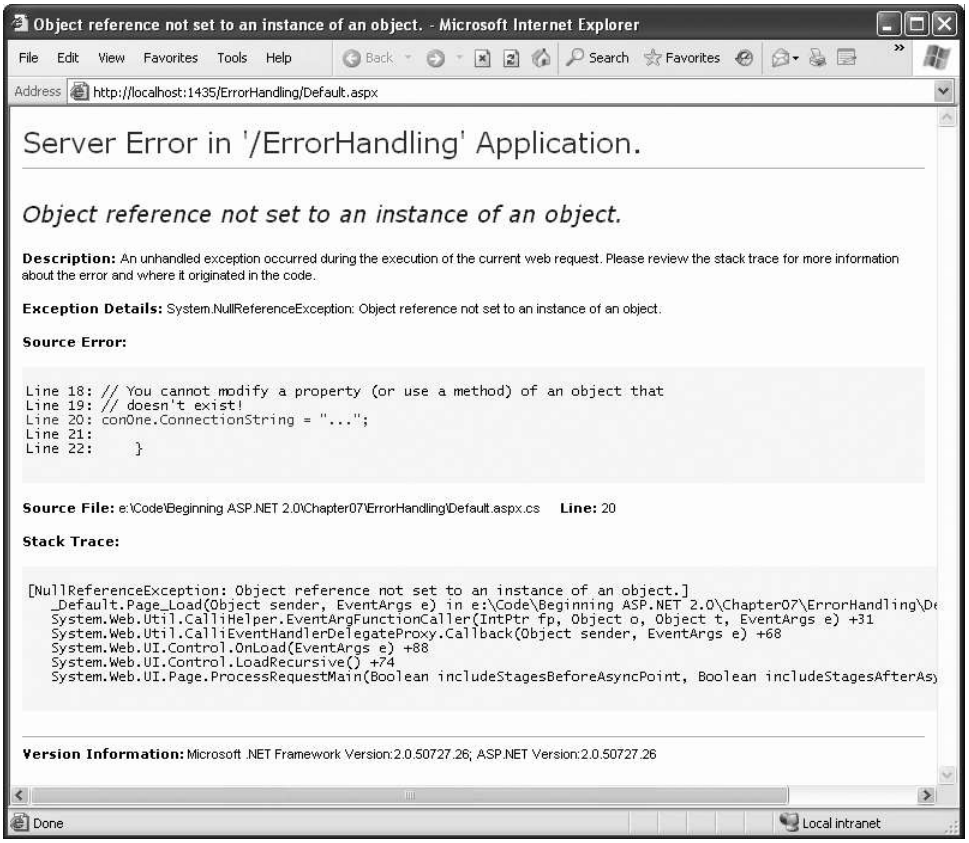
```
// Define a variable named conOne and create the object.
private SqlConnection conOne = new SqlConnection();

// Define a variable named conTwo, but don't create the object.
private SqlConnection conTwo;

public void cmdDoSomething_Click(object sender, EventArgs e)
{
```

Notes

```
// This works, because the object has been created
// with the new keyword.
conOne.ConnectionString = "...";
...

// The following statement will fail and generate a
// null reference exception.
// You cannot modify a property (or use a method) of an
// object that doesn't exist!
conTwo.ConnectionString = "...";
...
}
```

When an error occurs in your code, .NET checks to see whether any error handlers appear in the current scope. If the error occurs inside a method, .NET searches for local error handlers and then checks for any active error handlers in the calling code. If no error handlers are found, the page processing is aborted and an error page is displayed in the browser. Depending on whether the request is from the local computer or a remote client, the error page may show a detailed description (Figure 6.1) or a generic message.

Figure 6.1: A Sample Error Page



Notes

Even if an error is the result of invalid input or the failure of a third-party component, an error page can shatter the professional appearance of any application. The application users end up with a feeling that the application is unstable, insecure, or of poor quality and they're at least partially correct.

If an ASP.NET application is carefully designed and constructed, an error page will almost never appear. Errors may still occur because of unforeseen circumstances, but they will be caught in the code and identified. If the error is a critical one that the application cannot solve on its own, it will report a more useful (and user-friendly) page of information that might include a link to a support e-mail or a phone number where the customer can receive additional assistance.

6.2 Exception Handling

Most .NET languages support structured exception handling. Essentially, when an error occurs in your application, the .NET Framework creates an exception object that represents the problem. You can catch this object using an exception handler. If you fail to use an exception handler, your code will be aborted, and the user will see an error page.

Structured exception handling provides several key features:

1. ***Exceptions are object-based:*** Each exception provides a significant amount of diagnostic information wrapped into a neat object, instead of a simple message and an error code. These exception objects also support an InnerException property that allows you to wrap a generic error over the more specific error that caused it. You can even create and throw your own exception objects.
2. ***Exceptions are caught based on their type:*** This allows you to streamline error-handling code without needing to sift through obscure error codes.
3. ***Exception handlers use a modern block structure:*** This makes it easy to activate and deactivate different error handlers for different sections of code and handle their errors individually.
4. ***Exception handlers are multilayered:*** You can easily layer exception handlers on top of other exception handlers, some of which may check only for a specialized set of errors.
5. ***Exceptions are a generic part of the .NET Framework:*** This means they're completely crosslanguage compatible. Thus, a .NET component written in C# can throw an exception that you can catch in a web page written in VB.

Notes Exception handlers are a key programming technique. They allow you to react to problems that occur at runtime due to factors outside your control. However, you obviously shouldn't use exception handlers to hide the bugs that might crop up in your code! Instead, you need to track down these programmer mistakes at development time and correct them.

6.2.1 The Exception Class

Every exception class derives from the base class System.Exception. The .NET Framework is full of predefined exception classes, such as NullReferenceException, IOException, SqlException, and so on. The Exception class includes the essential functionality for identifying any type of error. Table 6.1 lists its most important members.

Notes

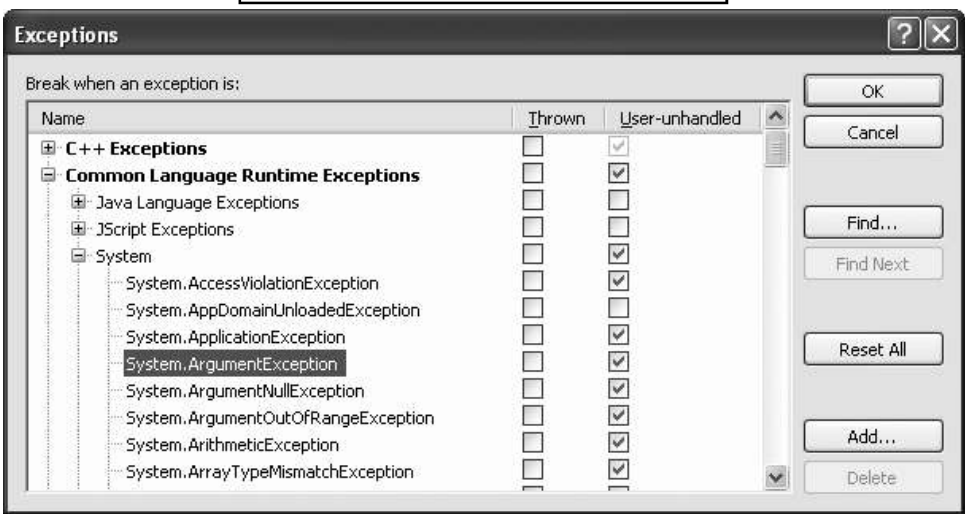
Table 6.1: Exception Properties

Member	Description
HelpLink	A link to a help document, which can be a relative or fully qualified URL (uniform resource locator) or URN (uniform resource name), such as file:///C:/ACME/MyApp/help.html#Err42. The .NET Framework doesn't use this property, but you can set it in your custom exceptions if you want to use it in your web page code.
InnerException	A nested exception. For example, a method might catch a simple file IO (input/output) error and create a higher-level "operation failed" error. The details about the original error could be retained in the InnerException property of the higher-level error.
Message	A text description with a significant amount of information describing the problem.
Source	The name of the application or object where the exception was raised.
StackTrace	A string that contains a list of all the current method calls on the stack, in order of most to least recent. This is useful for determining where the problem occurred.
TargetSite	A reflection object (an instance of the System.Reflection.MethodBase class) that provides some information about the method where the error occurred. This information includes generic method details such as the method name and the data types for its parameter and return values. It doesn't contain any information about the actual parameter values that were used when the problem occurred.
GetBaseException()	A method useful for nested exceptions that may have more than one layer. It retrieves the original (deepest nested) exception by moving to the base of the InnerException chain.

When you catch an exception in an ASP.NET page, it won't be an instance of the generic System.Exception class. Instead, it will be an object that represents a specific type of error. This object will be based on one of the many classes that inherit from System.Exception. These include diverse classes such as DivideByZeroException, ArithmeticException, IOException, SecurityException, and many more. Some of these classes provide additional details about the error in additional properties.

Task In ASP programming where you can use IOException function.

Figure 6.2: Visual Studio's Exception Viewer



Notes

Visual Studio provides a useful tool to browse through the exceptions in the .NET class library. Simply select Debug → Exceptions from the menu (you'll need to have a project open in order for this to work). The Exceptions dialog box will appear. Expand the Common Language Runtime Exceptions group, which shows a hierarchical tree of .NET exceptions arranged by namespace (Figure 6.2).

The Exceptions dialog box allows you to specify what exceptions should be handled by your code when debugging and what exceptions will cause Visual Studio to enter break mode immediately. That means you don't need to disable your error-handling code to troubleshoot a problem.

Example: You could choose to allow your program to handle a common FileNotFoundException (which could be caused by an invalid user selection) but instruct Visual Studio to pause execution if an unexpected DivideByZero exception occurs.

To set this up, add a check mark in the Thrown column next to the entry for the System. DivideByZero exception. This way, you'll be alerted as soon as the problem occurs. If you don't add a check mark to the Thrown column, your code will continue, run any exception handlers it has defined, and try to deal with the problem. You'll be notified only if an error occurs and no suitable exception handler is available.

6.2.2 The Exception Chain

Figure 6.3: Exceptions can be chained together

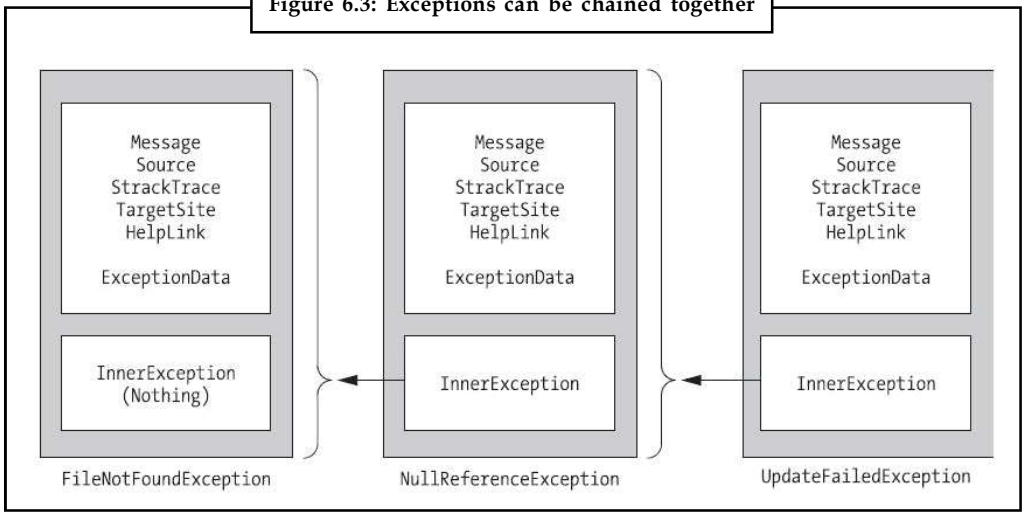


Figure 6.3 shows how the InnerException property works. In the specific scenario shown here, a FileNotFoundException led to a NullReferenceException, which led to a custom UpdateFailedException. Using an exception-handling block, the application can catch the UpdateFailedException. It can then get more information about the source of the problem by following the InnerException property to the NullReferenceException, which in turn references the original FileNotFoundException.

The InnerException property is an extremely useful tool for component-based programming. Generally, it's not much help if a component reports a low-level problem such as a null reference or a divide-by-zero error. Instead, it needs to communicate a more detailed message about which operation failed and what input may have been invalid. The calling code can then often correct the problem and retry the operation.

Notes

On the other hand, sometimes you're debugging a bug that lurks deep inside the component itself. In this case, you need to know precisely what caused the error you don't want to replace it with a higher-level exception that could obscure the root problem. Using an exception chain handles both these scenarios: you receive as many linked exception objects as needed, which can specify information from the least to the most specific error condition.

6.3 Handling Exceptions

The first line of defense in an application is to check for potential error conditions before performing an operation. For example, a program can explicitly check whether the divisor is 0 before performing a calculation, or if a file exists before attempting to open it:

```
if (divisor != 0)
{
// It's safe to divide some number by divisor.
}
if (System.IO.File.Exists("myfile.txt"))
{
// You can now open the myfile.txt file.
// However, you should still use exception handling because a variety of
// problems can intervene (insufficient rights, hardware failure, etc.).
}
```

Even if you perform this basic level of "quality assurance," your application is still vulnerable.

Example: You have no way to protect against all the possible file access problems that occur, including hardware failures or network problems that could arise spontaneously in the middle of an operation. Similarly, you have no way to validate a user ID and password for a database before attempting to open a connection and even if you did, that technique would be subject to its own set of potential errors. In some cases, it may not be practical to perform the full range of defensive checks, because they may impose a noticeable performance drag on your application. For all these reasons, you need a way to detect and deal with errors when they occur.

The solution is structured exception handling. To use structured exception handling, you wrap potentially problematic code in the special block structure shown here:

```
try
{
// Risky code goes here (i.e., opening a file, connecting to a database).
}
catch
{
// An error has been detected. You can deal with it here.
}
finally
{
// Time to clean up, regardless of whether or not there was an error.
}
```

Notes

The try statement enables error handling. Any exceptions that occur in the following lines can be "caught" automatically. The code in the catch block will be executed when an error is detected. And either way, whether a bug occurs or not, the finally section of the code will be executed last. This allows you to perform some basic cleanup, such as closing a database connection. The finally code is important because it will execute even if an error has occurred that will prevent the program from continuing. In other words, if an unrecoverable exception halts your application, you'll still have the chance to release resources.

The act of catching an exception neutralizes it. If all you want to do is render a specific error harmless, you don't even need to add any code in the catch block of your error handler. Usually, however, this portion of the code will be used to report the error to the user or log it for future reference. In a separate component (such as a business object), this code might handle the exception, perform some cleanup, and then rethrow it to the calling code, which will be in the best position to remedy it or alert the user. Or it might actually create a new exception object with additional information and throw that.

6.3.1 Catching Specific Exceptions

Structured exception handling is particularly flexible because it allows you to catch specific types of exceptions. To do so, you add multiple catch statements, each one identifying the type of exception (and providing a new variable to catch it in), as follows:

```
try
{
// Risky database code goes here.
}
catch (System.Data.SqlClient.SqlException err)
{
// Catches common problems like connection errors.
}
catch (System.NullReferenceException err)
{
// Catches problems resulting from an uninitialized object.
}
```

An exception will be caught as long as it's an instance of the indicated class or if it's derived from that class. In other words, if you use this statement:

```
catch (Exception err)
```

You will catch any exception, because every exception object is derived from the System.Exception base class.

Exception blocks work a little like conditional code. As soon as a matching exception handler is found, the appropriate catch code is invoked. Therefore, you must organize your catch statements from most specific to least specific:

```
try
{
// Risky database code goes here.
}
catch (System.Data.SqlClient.SqlException err)
```

Notes

```
{
// Catches common problems like connection errors.
}
catch (System.NullReferenceException err)
{
// Catches problems resulting from an uninitialized object.
}
catch (System.Exception err)
{
// Catches any other errors.
}
```

Ending with a catch statement for the base Exception class is often a good idea to make sure no errors slip through. However, in component-based programming, you should make sure you intercept only those exceptions you can deal with or recover from. Otherwise, it's better to let the calling code catch the original error.

6.3.2 Nested Exception Handlers

When an exception is thrown, .NET tries to find a matching catch statement in the current method. If the code isn't in a local structured exception block, or if none of the catch statements match the exception, .NET will move up the call stack one level at a time, searching for active exception handlers.

Consider the example shown here, where the Page.Load event handler calls a private DivideNumbers() method:

```
protected void Page_Load(Object sender, EventArgs e)
{
try
{
DivideNumbers(5, 0);
}
catch (DivideByZeroException err)
{
// Report error here.
}
}
private decimal DivideNumbers(decimal number, decimal divisor)
{
return number/divisor;
}
```

In this example, the DivideNumbers() method lacks any sort of exception handler. However, the DivideNumbers() method call is made inside a try block, which means the problem will be caught further upstream in the calling code. This is a good approach because the DivideNumbers() routine could be used in a variety of circumstances (or if it's part of a component, in a variety of different types of applications). It really has no access to any kind of user interface and can't directly report an error. Only the calling code is in a position to determine whether the problem is a serious one or a minor one, and only the calling code can prompt the user for more information or report error details in the web page.

Notes

Notes Great care is taken to use the decimal data type rather than the more common double data type. That's because contrary to what you might expect, it is acceptable to divide a double by 0. The result is the special value Double.PositiveInfinity (or Double.NegativeInfinity if you divide a negative number by 0).

You can also overlap exception handlers in such a way that different exception handlers filter out different types of problems. Here's one such example:

```
protected void Page_Load(Object sender, EventArgs e)
{
try
{
decimal average = GetAverageCost(DateTime.Now);
}
catch (DivideByZeroException err)
{
// Report error here.
}
}
private decimal GetAverageCost(DateTime saleDate)
{
try
{
// Use Database access code here to retrieve all the sale records
// for this date, and calculate the average.
}
catch (SqlException err)
{
// Handle a database related problem.
}
finally
{
// Close the database connection.
}
}
```

Dissecting the Code . . .

You should be aware of the following points:

1. If an SqlException occurs during the database operation, it will be caught in the GetAverageCost() method.
2. If a DivideByZeroException occurs (for example, the method receives no records but still attempts to calculate an average), the exception will be caught in the calling Page.Load event handler.
3. If another problem occurs (such as a null reference exception), no active exception handler exists to catch it. In this case, .NET will search through the entire call stack without finding

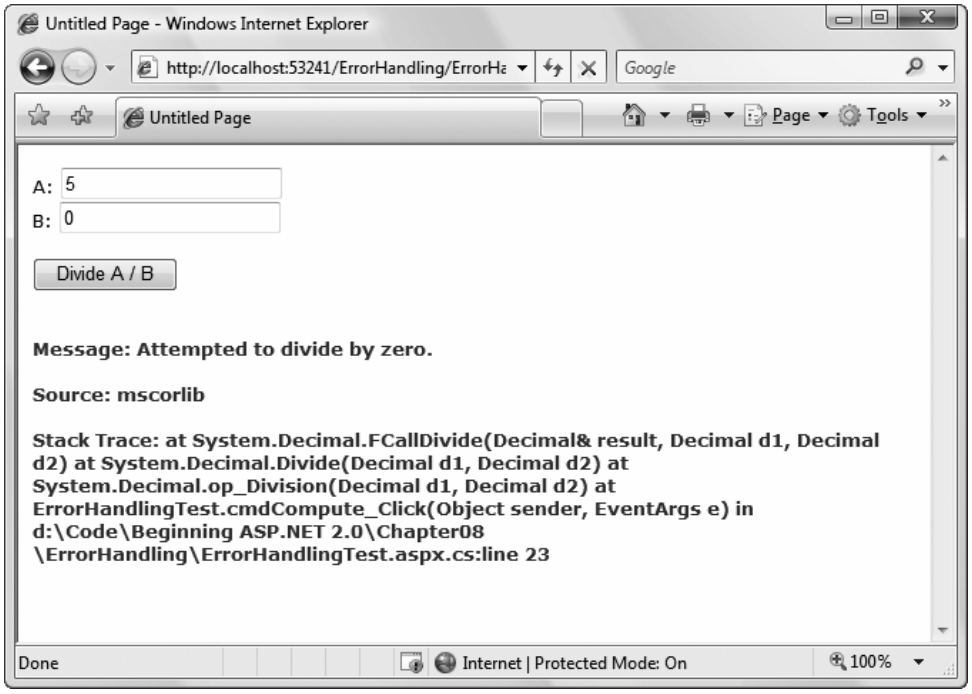
Notes

a matching catch statement in an active exception handler and will generate a runtime error, end the program, and return a page with exception information.

6.3.3 Exception Handling in Action

You can use a simple program to test exceptions and see what sort of information is retrieved. This program allows a user to enter two values and attempts to divide them. It then reports all the related exception information in the page (Figure 6.4).

Figure 6.4: Catching and Displaying Exception Information



Obviously, you can easily prevent this exception from occurring by using extra codesafety checks, or elegantly resolve it using the validation controls. However, this code provides a good example of how you can deal with the properties of an exception object. It also gives you a good idea about what sort of information will be returned.

Here's the page class code for this example:

```
public partial class ErrorHandlingTest : System.Web.UI.Page
{
protected void cmdCompute_Click(Object sender, EventArgs e)
{
try
{
decimal a, b, result;
a = Decimal.Parse(txtA.Text);
b = Decimal.Parse(txtB.Text);
result = a / b;
lblResult.Text = result.ToString();
lblResult.ForeColor = System.Drawing.Color.Black;
```

Notes

```
}
catch (Exception err)
{
lblResult.Text = "<b>Message:</b> " + err.Message;
lblResult.Text += "<br /><br />";
lblResult.Text += "<b>Source:</b> " + err.Source;
lblResult.Text += "<br /><br />";
lblResult.Text += "<b>Stack Trace:</b> " + err.StackTrace;
lblResult.ForeColor = System.Drawing.Color.Red;
}
}
}
```

Note that as soon as the error occurs, execution is transferred to an exception handler. The code in the try block isn't completed. It's for that reason that the result for the label is set in the try block. These lines will be executed only if the division code runs error-free.

Task Write a code to catch the exceptions in ASP page.

6.4 Throwing your own Exceptions

You can also define your own exception objects to represent custom error conditions. All you need to do is create an instance of the appropriate exception class and then use the throw statement.

The next example introduces a modified DivideNumbers() method. It explicitly checks whether the specified divisor is 0 and then manually creates and throws an instance of the DivideByZeroException class to indicate the problem, rather than attempt the operation. Depending on the code, this pattern can save time by eliminating some unnecessary steps, or it can prevent a task from being initiated if it can't be completed successfully.

```
protected void Page_Load(Object sender, EventArgs e)
{
try
{
DivideNumbers(5, 0);
}
catch (DivideByZeroException err)
{
// Report error here.
}
}
private decimal DivideNumbers(decimal number, decimal divisor)
{
if (divisor == 0)
{
DivideByZeroException err = new DivideByZeroException();
throw err;
```

Notes

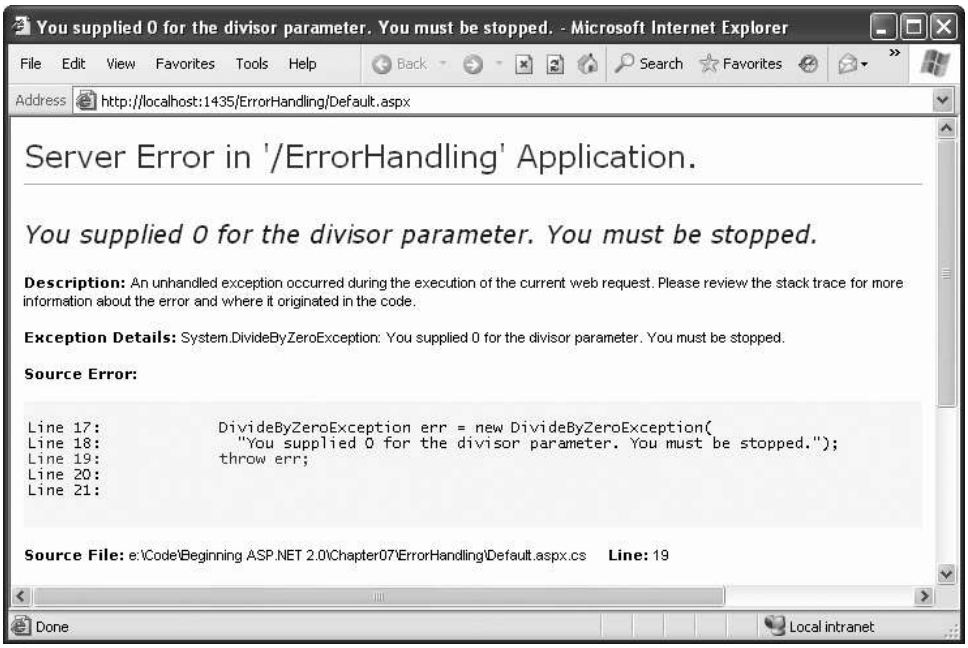
```
}
else
{
return number/divisor;
}
}
```

Alternatively, you can create a .NET exception object and specify a custom error message by using a different constructor:

```
private decimal DivideNumbers(decimal number, decimal divisor)
{
if (divisor == 0)
{
DivideByZeroException err = new DivideByZeroException(
"You supplied 0 for the divisor parameter. You must be stopped.");
throw err;
}
else
{
return number/divisor;
}
}
```

In this case, any ordinary exception handler will still catch the DivideByZeroException. The only difference is that the error object has a modified Message property that contains the custom string. Figure 6.5 shows the resulting exception.

Figure 6.5: Standard Exception and Custom Message



Notes

Throwing an exception is most useful in component-based programming. In componentbased programming, your ASP.NET page is creating objects and calling methods from a class defined in a separately compiled assembly. In this case, the class in the component needs to be able to notify the calling code (the web application) of any errors. The component should handle recoverable errors quietly and not pass them up to the calling code. On the other hand, if an unrecoverable error occurs, it should always be indicated with an exception and never through another mechanism (such as a return code).

If you can find an exception in the class library that accurately reflects the problem that has occurred, you should throw it. If you need to return additional or specialized information, you can create your own custom exception class.

Custom exception classes should always inherit from System.ApplicationException, which itself derives from the base Exception class. This allows .NET to distinguish between two broad classes of exceptions those you create and those that are native to the .NET Framework.

When you create an exception class, you can add properties to record additional information. For example, here is a special class that records information about the failed attempt to divide by zero:

```
public class CustomDivideByZeroException : ApplicationException
{
// Add a variable to specify the "other" number.
// This might help diagnose the problem.
public decimal DividingNumber;
}
```

You can throw this custom exception like this:

```
private decimal DivideNumbers(decimal number, decimal divisor)
{
if (divisor == 0)
{
CustomDivideByZeroException err = new CustomDivideByZeroException();
err.DividingNumber = number;
throw err;
}
else
{
return number/divisor;
}
}
```

To perfect the custom exception, you need to supply it with the three standard constructors. This allows your exception class to be created in the standard ways that every exception supports:

1. On its own, with no arguments
2. With a custom message
3. With a custom message and an exception object to use as the inner exception

Notes

These constructors don't actually need to contain any code. All these constructors need to do is forward the parameters to the base class (the constructors in the inherited Application-Exception class) using the base keyword, as shown here:

```
public class CustomDivideByZeroException : ApplicationException
{
// Add a variable to specify the "other" number.
private decimal dividingNumber;
public decimal DividingNumber
{
get { return dividingNumber; }
set { dividingNumber = value; }
}
public CustomDivideByZeroException() : base()
{}
public CustomDivideByZeroException(string message) : base(message)
{}
public CustomDivideByZeroException(string message, Exception inner) :
base(message, inner)
{}
}
```

The third constructor is particularly useful for component programming. It allows you to set the InnerException property with the exception object that caused the original problem. The next example shows how you could use this constructor with a component class called ArithmeticUtility:

```
public class ArithmeticUtilityException : ApplicationException
{
public ArithmeticUtilityException() : base()
{}
public ArithmeticUtilityException(string message) : base(message)
{}
public ArithmeticUtilityException(string message, Exception inner) :
base(message, inner)
{}
}
public class ArithmeticUtility
{
private decimal Divide(decimal number, decimal divisor)
{
try
{
return number/divisor;
}
catch (Exception err)
{
// Create an instance of the specialized exception class,
```

Notes

```
// and place the original error in the InnerException property.
ArithmeticUtilityException errNew =
new ArithmeticUtilityException("Divide by zero", err);
// Now throw the new exception.
throw errNew;
}
}
}
```

Remember, custom exception classes are really just a standardized way for one class to communicate an error to a different portion of code. If you aren't using components or your own utility classes, you probably don't need to create custom exception classes.

6.5 Logging Exceptions

In many cases, it's best not only to detect and catch exceptions but to log them as well.

Example: Some problems may occur only when your web server is dealing with a particularly large load. Other problems might recur intermittently, with no obvious causes. To diagnose these errors and build a larger picture of site problems, you need to log exceptions so they can be reviewed later.

The .NET Framework provides a wide range of logging tools. When certain errors occur, you can send an e-mail, add a database record, or create and write to a file. We describe many of these techniques in other parts of this book. However, you should keep your logging code as simple as possible. For example, you'll probably run into trouble if you try to log a database exception using another table in the database.

One of the most fail-safe logging tools is the Windows event logging system, which is built into the Windows operating system and available to any application. Using the Windows event logs, your website can write text messages that record errors or unusual events. The Windows event logs store your messages as well as various other details, such as the message type (information, error, and so on) and the time the message was left.

6.5.1 Viewing the Windows Event Logs

To view the Windows event logs, you use the Event Viewer tool that's included with Windows. To launch it, begin by selecting Start → Settings → Control Panel (or just Start → Control Panel in Windows Vista). Open the Administrative Tools group, and then choose Event Viewer.

If you're running Windows XP or Windows Server 2003, you'll see just three logs Application, Security, and System. If you're running Windows Vista or Windows Server 2008, you'll find these three plus a Setup log, all of which appear under the Windows Logs section (Figure 6.6). Table 6.2 describes these standard Windows logs.

Notes

Figure 6.6: The Event Viewer

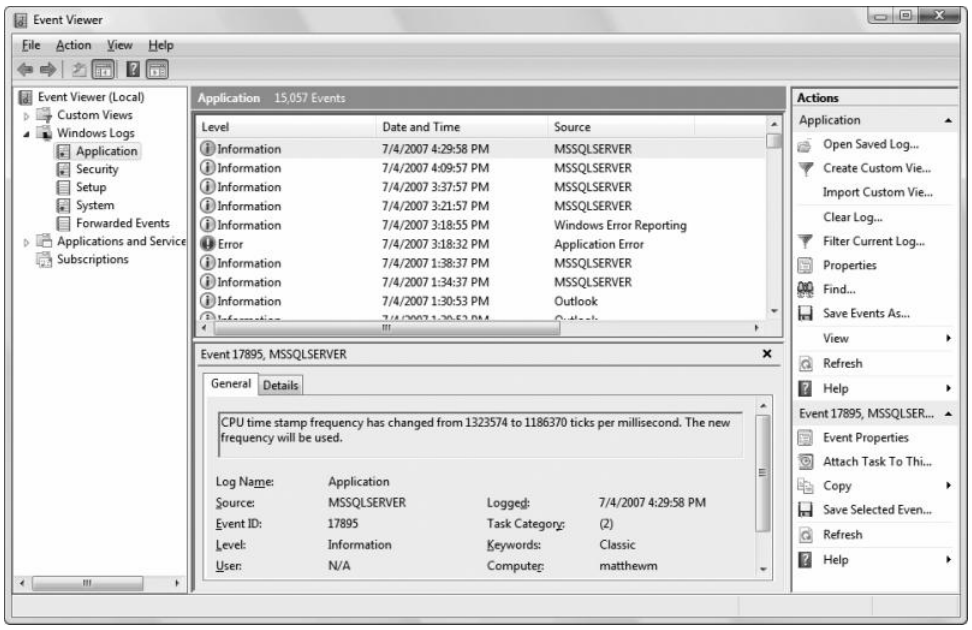


Table 6.2: Windows Event Logs

LogName	Description
Application	Used to track errors or notifications from any application. Generally, you'll use this log when you're performing event logging, or you'll create your own custom log.
Security	Used to track security-related problems but generally used exclusively by the operating system.
System	Used to track operating system events.
Setup	Used to track issues that occur when installing Windows updates or other software. This log only appears in Windows Vista.

Using the Event Viewer, you can perform a variety of management tasks with the logs.

Example: If you right-click one of the logs in the Event Viewer list you'll see options that allow you to clear the events in the log, save the log entries to another file, and import an external log file.

Each event record in an event log identifies the source (generally, the application or service that created the record), the type of notification (error, information, warning), and the time the log entry was inserted. In Windows Vista, you simply need to select a log entry and its information will appear in a display area underneath the list of entries (Figure 6.6). In Windows XP or Windows Server 2003, you need to double-click a log entry to see the full information.

You can also review event logs in Visual Studio. First, display the Server Explorer window (if it's not already visible) by choosing View → Server Explorer. (The Server Explorer window usually appears at the left side of the Visual Studio window, where it shares space with the Toolbox.) Using the Server Explorer, expand the Servers → [ComputerName] → Event Logs group to see a list of event logs on your computer. This list is a bit longer than what you saw in the Event Viewer, because it includes both the Windows event logs you saw and custom event logs for specific applications.

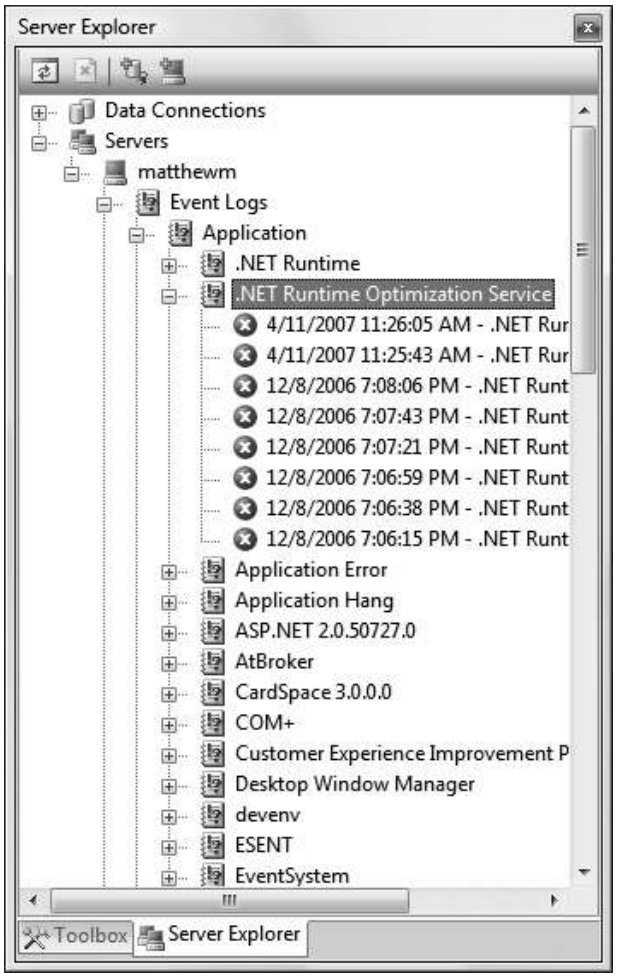
Notes

If you expand an event log in the Server Explorer window, you'll find all the event log entries, grouped according to the source that made the log entry. Figure 6.7 shows some of the event logs left in the Application log on the current computer by the event source .NET Runtime Optimization Source. Once you select a log entry, you can view its specific details (such as the event log message and the time it was left) in the Properties window.

One of the potential problems with event logs is that old entries are automatically discarded when the event log reaches a maximum size. InWindows XP, the default log size is a stingy 0.5MB, although log entries are kept for at least 7 days even if they exceed this size limit (unless you specify otherwise). In Windows Vista, logs get a much more reasonable upper limit of 20MB (except for the new Setup log, which gets just 1MB).

No matter which operating system you're using, you'll find that logs grow quickly. That means that unless you're using a custom event log that has lots of space, your log entries might not last for a long period of time. Ideally, you should use event logs to record information that is reviewed and acted on over a relatively short period of time. For example, event logs are a good choice if you plan to log application errors and review them to diagnose strange behavior immediately after it happens. Event logs don't make as much sense if you want to get a detailed picture of application activity six months later, because Windows (or someone else) may delete old log entries. In this scenario, a custom database makes more sense.

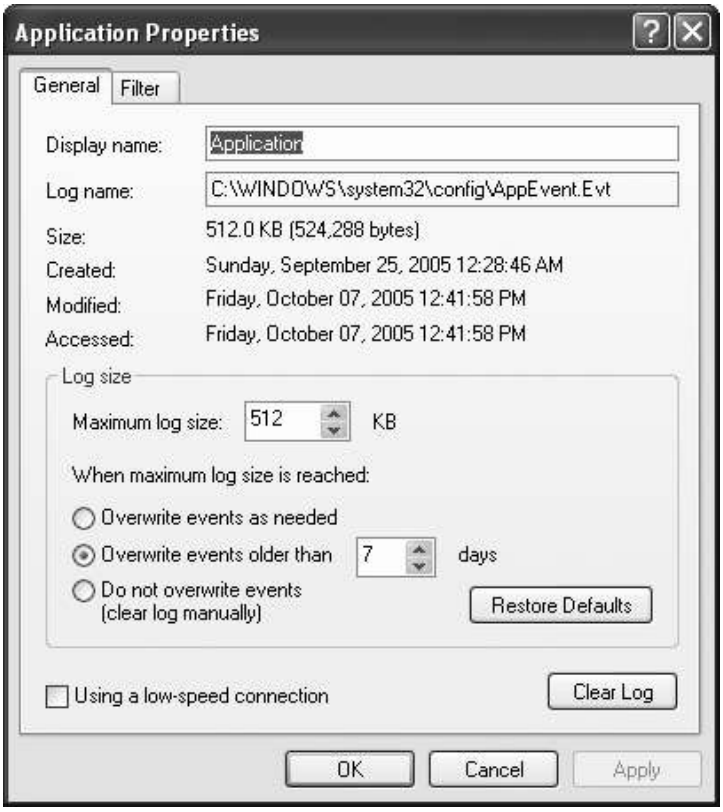
Figure 6.7: Viewing event log entries in Visual Studio



Notes

If you want to add a little more breathing room to an existing log, you can change its maximum size. This is a particularly worthwhile step if you plan to use the application log in Windows XP. To do so, right-click the log and choose Properties. You'll see the Application Properties window shown in Figure 6.8, where you can change the maximum size.

Figure 6.8: Log Properties



6.5.2 Writing to the Event Log

You can interact with event logs in an ASP.NET page by using the classes in the System.Diagnostics namespace. First, import the namespace at the beginning of your code-behind file:

```
using System.Diagnostics;
```

The following example rewrites the simple ErrorTest page to use event logging:

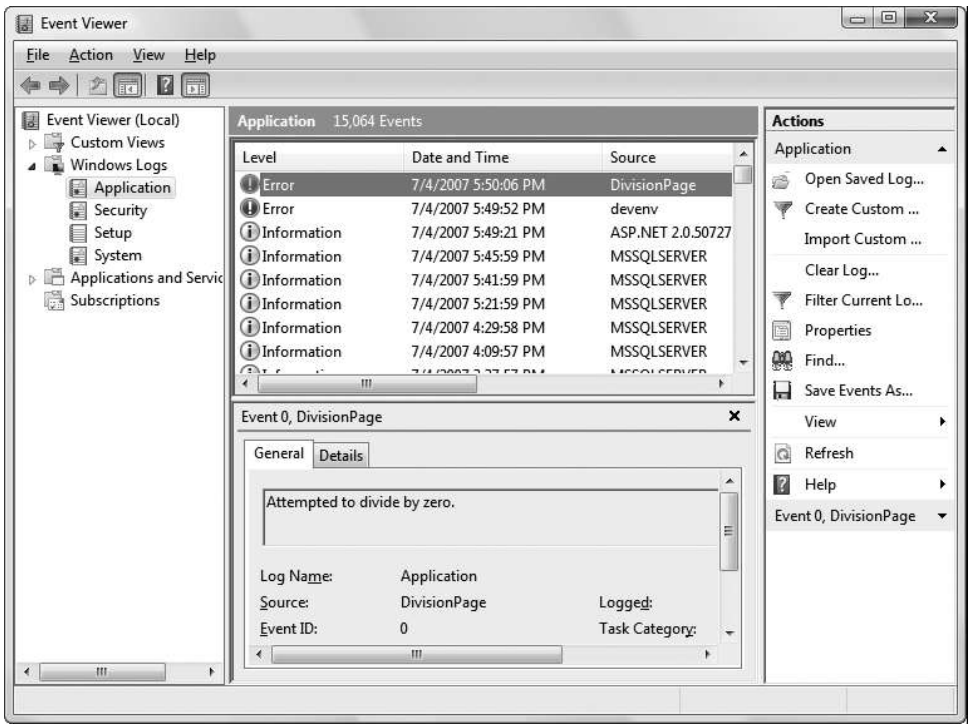
```
public partial class ErrorTestLog : Page
{
protected void cmdCompute_Click(Object sender, EventArgs e)
{
try
{
decimal a, b, result;
a = Decimal.Parse(txtA.Text);
b = Decimal.Parse(txtB.Text);
result = a / b;
```

Notes

```
lblResult.Text = result.ToString();
lblResult.ForeColor = System.Drawing.Color.Black;
}
catch (Exception err)
{
lblResult.Text = "<b>Message:</b> " + err.Message + "<br /><br />";
lblResult.Text += "<b>Source:</b> " + err.Source + "<br /><br />";
lblResult.Text += "<b>Stack Trace:</b> " + err.StackTrace;
lblResult.ForeColor = System.Drawing.Color.Red;

// Write the information to the event log.
EventLog log = new EventLog();
log.Source = "DivisionPage";
log.WriteEntry(err.Message, EventLogEntryType.Error);
}
}
}
```

The event log record will now appear in the Event Viewer utility, as shown in Figure 6.9. Note that logging is intended for the system administrator or developer. It doesn't replace the code you use to notify the user and explain that a problem has occurred.

Figure 6.9: An Event Record



Notes

6.5.3 Custom Logs

You can also log errors to a custom log. For example, you could create a log with your company name and add records to it for all your ASP.NET applications. You might even want to create an individual log for a particularly large application and use the Source property of each entry to indicate the page (or web service method) that caused the problem.

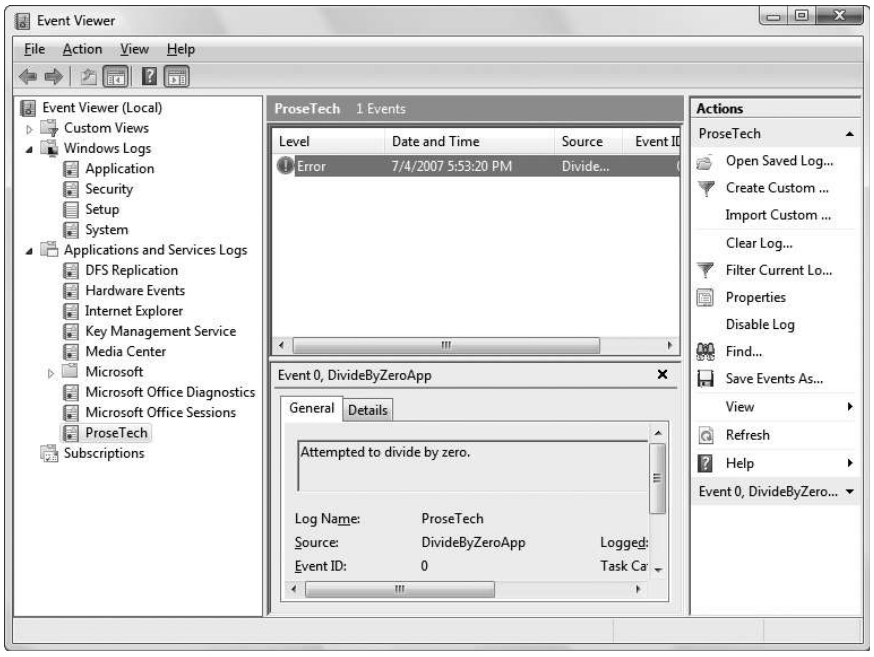
Accessing a custom log is easy you just need to use a different constructor for the EventLog class to specify the custom log name. You also need to register an event source for the log. This initial step needs to be performed only once in fact, you'll receive an error if you try to create the same event source. Typically, you'll use the name of the application as the event source.

Here's an example that uses a custom log named ProseTech and registers the event source DivideByZeroApp:

```
// Register the event source if needed.
if (!EventLog.SourceExists("ProseTech"))
{
// This registers the event source and creates the custom log,
// if needed.
EventLog.CreateEventSource("DivideByZeroApp", "ProseTech");
}
// Open the log. If the log doesn't exist,
// it will be created automatically.
EventLog log = new EventLog("ProseTech");
log.Source = "DivideByZeroApp";
log.WriteEntry(err.Message, EventLogEntryType.Error);
```

If you specify the name of a log that doesn't exist when you use the CreateEventSource() method, the system will create a new, custom event log for you the first time you write an entry.

Figure 6.10: A Custom Log



Notes

In order to see a newly created event log in the Event Viewer tool, you'll need to exit Event Viewer and restart it. In Windows XP and Windows Server 2003, custom event logs appear alongside the standard Windows event logs. In Windows Vista, they appear in a separate group named Applications and Services Logs, as shown in Figure 6.10.

You can use this sort of code anywhere in your application. Usually, you'll use logging code when responding to an exception that might be a symptom of a deeper problem.

6.5.4 A Custom Logging Class

Rather than adding too much logging code in the catch block, a better approach is to create a separate class that handles the event logging grunt work. You can then use that class from any web page, without duplicating any code.

To use this approach, begin by creating a new code file in the App_Code subfolder of your website. You can do this in Visual Studio by choosing Website → Add New Item. In the Add New Item dialog box, choose Class, pick a suitable file name, and then click Add.

Here's an example of a class named MyLogger that handles the event logging details:

```
public class MyLogger
{
public void LogError(string pageInError, Exception err)
{
RegisterLog();
EventLog log = new EventLog("ProseTech");
log.Source = pageInError;
log.WriteEntry(err.Message, EventLogEntryType.Error);
}
private void RegisterLog()
{
// Register the event source if needed.
if (!EventLog.SourceExists("ProseTech"))
{
EventLog.CreateEventSource("DivideByZeroApp", "ProseTech");
}
}
}
```

Once you have a class in the App_Code folder, it's easy to use it anywhere in your website. Here's how you might use the MyLogger class in a web page to log an exception:

```
try
{
// Risky code goes here.
}
catch (Exception err)
{
// Log the error using the logging class.
MyLogger logger = new MyLogger();
logger.LogError(Request.Path, err);
```

Notes

```
// Now handle the error as appropriate for this page.
lblResult.Text = "Sorry. An error occurred.";
}
```

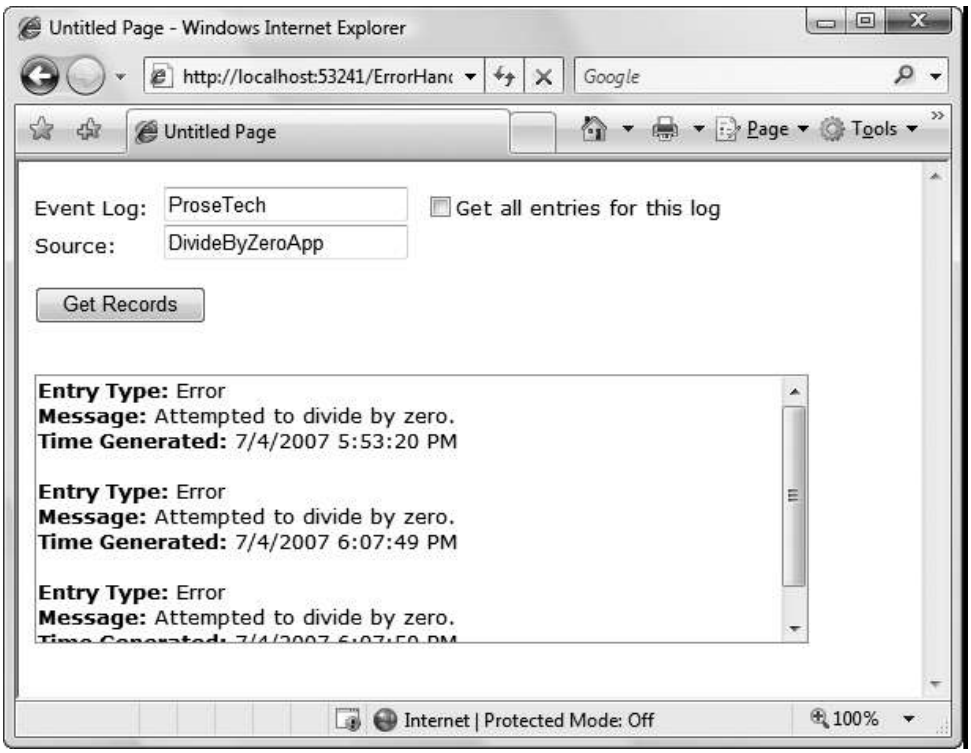
If you write log entries frequently, you may not want to check if the log exists every time you want to write an entry. Instead, you could create the event source once when the application first starts up using an application event handler in the Global.asax file.

6.5.5 Retrieving Log Information

One of the disadvantages of the event logs is that they're tied to the web server. This can make it difficult to review log entries if you don't have a way to access the server (although you can read them from another computer on the same network). This problem has several possible solutions. One interesting technique involves using a special administration page. This SP.NET page can use the EventLog class to retrieve and display all the information from the event log.

Figure 6.11 shows in a simple web page all the entries that were left by the ErrorTestCustomLog page. The results are shown using a label in a scrollable panel (a Panel control with the Scrollbars property set to Vertical).

Figure 6.11: A Log Viewer Page



Here's the web page code you'll need:

```
public partial class EventReviewPage : System.Web.UI.Page
{
protected void cmdGet_Click(Object sender, EventArgs e)
{
lblResult.Text = "";
```

Notes

```
// Check if the log exists.
if (!EventLog.Exists(txtLog.Text))
{
lblResult.Text = "The event log " + txtLog.Text ;
lblResult.Text += " doesn't exist.";
}
else
{
EventLog log = new EventLog(txtLog.Text);
foreach (EventLogEntry entry in log.Entries)
{
// Write the event entries to the page.
if (chkAll.Checked ||
entry.Source == txtSource.Text)
{
lblResult.Text += "<b>Entry Type:</b> ";
lblResult.Text += entry.EntryType.ToString();
lblResult.Text += "<br /><b>Message:</b> ";
lblResult.Text += entry.Message;
lblResult.Text += "<br /><b>Time Generated:</b> ";
lblResult.Text += entry.TimeGenerated;
lblResult.Text += "<br /><br />";
}
}
}
}

protected void chkAll_CheckedChanged(Object sender,
EventArgs e)
{
// The chkAll control has AutoPostback = true.
if (chkAll.Checked)
{
txtSource.Text = "";
txtSource.Enabled = false;
}
else
{
txtSource.Enabled = true;
}
}
}
```

If you choose to display all the entries from the application log, the page will perform slowly. Two factors are at work here. First, it takes time to retrieve each event log entry; a typical application log can easily hold several thousand entries. Second, the code used to append text to

Notes

the Label control is inefficient. Every time you add a new piece of information to the Label.Text property, .NET needs to generate a new String object. A better solution is to use the specialized System.Text.StringBuilder class, which is designed to handle intensive string processing with a lower overhead by managing an internal buffer or memory.

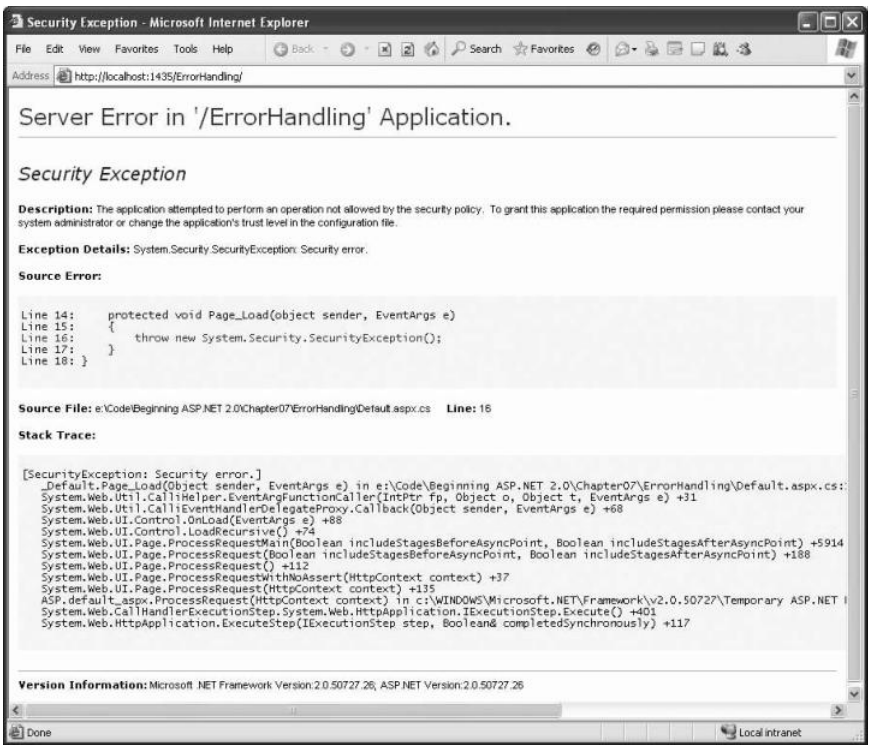
```
// For maximum performance, join all the event
// information into one large string using the
// StringBuilder.
System.Text.StringBuilder sb = new System.Text.StringBuilder();

EventLog log = new EventLog(txtLog.Text);
foreach (EventLogEntry entry in log.Entries)
{
// Write the event entries to the StringBuilder.
if (chkAll.Checked ||
entry.Source == txtSource.Text)
{
sb.Append("<b>Entry Type:</b> ");
sb.Append(entry.EntryType.ToString());
sb.Append("<br /><b>Message:</b> ");
sb.Append(entry.Message);
sb.Append("<br /><b>Time Generated:</b> ");
sb.Append(entry.TimeGenerated);
sb.Append("<br /><br />");
}
// Copy the complete text to the web page.
lblResult.Text = sb.ToString();
}
```

6.6 Error Pages

As you create and test an ASP.NET application, you'll become familiar with the rich error pages that are shown to describe unhandled errors. These rich error pages are extremely useful for diagnosing problems during development, because they contain a wealth of information. Some of this information includes the source code where the problem occurred (with the offending line highlighted), the type of error, and a detailed error message describing the problem. Figure 6.12 shows a sample rich error page.

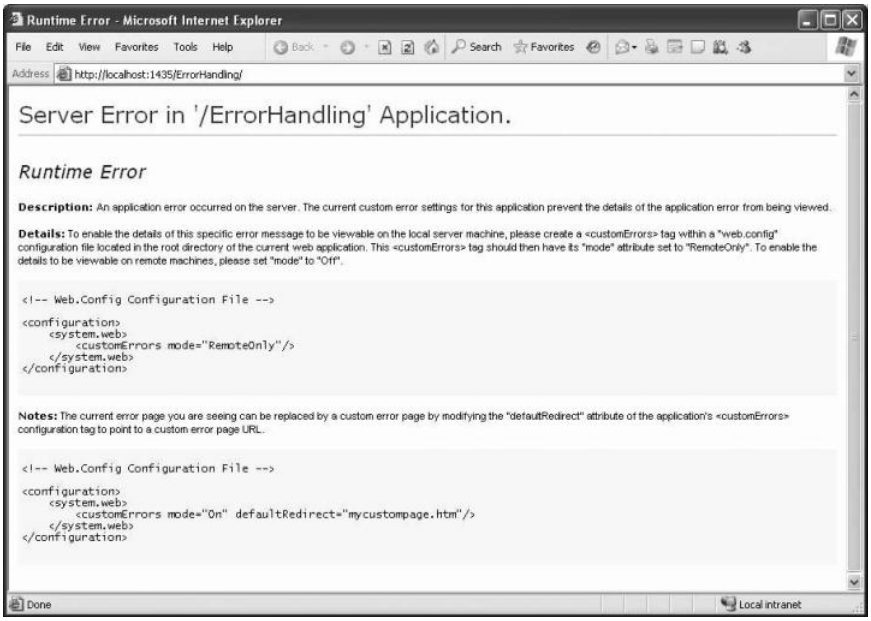
Notes

Figure 6.12: A Rich ASP.NET Error Page



By default, this error page is shown only for local requests that are sent from the http://localhost domain. (This domain always refers to the current computer, regardless of its actual server name or Internet address.) ASP.NET doesn't show rich error pages for requests from other computers; they receive the rather unhelpful generic page shown in Figure 6.13.

Figure 6.13: A Generic Client Error Page



Notes

This generic page lacks any specific details about the type of error or the offending code. Sharing that information with end users would be a security risk (potentially exposing sensitive details about the source code), and it would be completely unhelpful, because clients are never in a position to modify the source code themselves. Instead, the page includes a generic message explaining that an error has occurred and describing how to change the configuration settings (by modifying the web.config file) so that remote users also see the rich error pages.

6.6.1 Error Modes

You can change the configuration of your web application so that all users see the rich ASP.NET error pages with detailed error information. This option is intended as a testing tool. For example, in the initial rollout of an application beta, you might use field testers. These field testers would need to report specific information about application errors to aid in the debugging process. Similarly, you could use remote error pages if you're working with a team of developers and testing an ASP.NET application from a server on your local network. In both of these situations, the web application is uploaded to a remote computer before you begin testing it.

To change the error mode, you need to add the <customErrors> section to the web.config file. Here it is, with the default setting:

```
<configuration>
<system.web>
<customErrors mode="RemoteOnly" />
...
</system.web>
...
</configuration>
```

Table 6.3 lists the options for the mode attribute.

Table 6.3: Error Modes

Error Mode	Description
RemoteOnly	Generic error pages are shown for remote users. Rich error pages are shown for local requests (requests that are made from the current computer). This is the default setting.
Off	Rich error pages are shown for all users, regardless of the source of the request. This setting is helpful in many development scenarios but should not be used in a deployed application. (Not only will the rich error pages confuse users, but they may reveal sensitive information about your code.)
On	Generic error pages are shown for all users, regardless of the source of the request. This is the most secure option, but it complicates debugging because you'll need logging or tracing code to report error information.

It makes good sense to hide the rich error pages from ordinary users. However, the generic error pages really aren't that much more useful. The message they show has less information and won't reveal any secrets about your code (Figure 6.13), but it's still confusing for mere mortals. ASP.NET allows you to replace the generic error page with a custom error page of your own devising.

6.6.2 Custom Error Pages

In a deployed application, you should use the On or RemoteOnly error mode. Any errors in your application should be dealt with through error-handling code, which can then present a

Notes

helpful and user-oriented message (rather than the developer-oriented code details in ASP.NET's rich error messages).

However, you can't catch every possible error in an ASP.NET application. For example, a hardware failure could occur spontaneously in the middle of an ordinary code statement that could not normally cause an error. More commonly, the user might encounter an HTTP error by requesting a page that doesn't exist. ASP.NET allows you to handle these problems with custom error pages.

You can implement custom error pages in two ways. You can create a single generic error page and configure ASP.NET to use it by modifying the web.config file as shown here:

```
<configuration>
<system.web>
<customErrors mode="RemoteOnly" defaultRedirect="DefaultError.aspx" />
</system.web>
</configuration>
```

ASP.NET will now exhibit the following behavior:

If ASP.NET encounters an HTTP error while serving the request, it will forward the user to the DefaultError.aspx web page.

If ASP.NET encounters an unhandled application error and the mode is set to On (Table 6.3), it will forward the user to the DefaultError.aspx. Remote users will never see the generic ASP.NET error page.

If ASP.NET encounters an unhandled application error and the mode is set to Off, it will display the ASP.NET error page instead.

If ASP.NET encounters an unhandled application error and the mode is set to RemoteOnly, the behavior depends on where the request is originating from. If it's a local request being made from the same computer, you'll get the ASP.NET error page with the diagnostic information. Otherwise, you'll see the DefaultError.aspx page.

Notes What happens if an error occurs in the error page itself? In a custom error page (in this case, DefaultError.aspx), ASP.NET will not be able to handle an error. It will not try to reforward the user to the same page. Instead, it will display the normal client error page with the generic message.

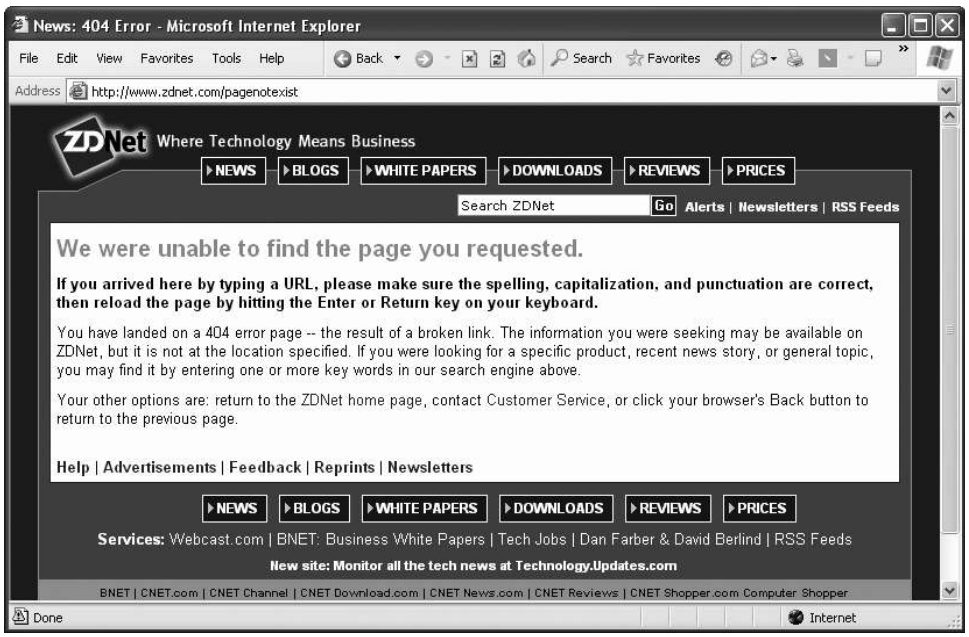
You can also create error pages targeted at specific types of HTTP errors (such as the infamous 404 Not Found error or Access Denied). This technique is commonly used with websites to provide friendly equivalents for common problems. Figure 6.14 shows how one site handles this issue.

To define an error-specific custom page, you add an <error> element to the <customErrors> element. The <error> element identifies the HTTP error code and the redirect page.

```
<configuration>
<system.web>
<customErrors defaultRedirect="DefaultError.aspx">
<error statusCode="404" redirect="404.aspx" />
<customErrors>
</system.web>
</configuration>
```

Notes

Figure 6.14: A Sample Custom Error Page



In this example, the user will be redirected to the 404.aspx page when requesting an ASP.NET page that doesn't exist. This custom error page may not work exactly the way you expect, because it comes into effect only if ASP.NET is handling the request.

Example: If you request the nonexistent page whateverpage.aspx, you'll be redirected to 404.aspx, because the .aspx file extension is registered to the ASP.NET service. However, if you request the nonexistent page whateverpage.html, ASP.NET will not process the request, and the default redirect setting specified in IIS will be used.

When an error occurs that isn't specifically addressed by a custom <error> element, the default error page will be used.

Task How will you configure error page? Discuss.

6.7 Page Tracing

ASP.NET's detailed error pages are extremely helpful when you're testing and perfecting an application. However, sometimes you need more information to verify that your application is performing properly or to track down logic errors, which may produce invalid data but no obvious exceptions.

You could try to catch these errors by recording diagnostic information in an event log, but this assumes that someone will actually review the log regularly. More aggressively, you could display some information directly in the web page. The problem with this strategy is that you need to remove (or at least comment out) all this extra code before you deploy your web application. Otherwise, your website users could see strange debugging messages when they least expect it.

Notes

Fortunately, there's an easier way to solve the problem without resorting to a homegrown solution. ASP.NET provides a feature called tracing that gives you a far more convenient and flexible way to report diagnostic information.

6.7.1 Enabling Tracing

To use tracing, you need to explicitly enable it. There are several ways to switch on tracing. One of the easiest ways is by adding an attribute to the Page directive in the .aspx file:

```
<%@ Page Trace="true" ... %>
```

You can also enable tracing using the built-in Trace object (which is an instance of the System.Web.TraceContext class). Here's an example of how you might turn tracing on in the Page.Load event handler:

```
protected void Page_Load(Object sender, EventArgs e)
{
Trace.IsEnabled = true;
}
```

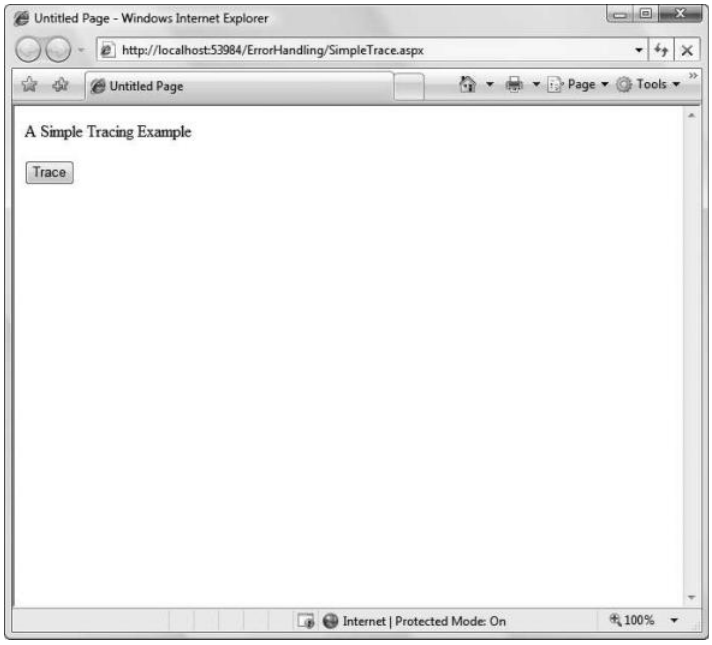
This technique is useful because it allows you to enable or disable tracing for a page under specific circumstances that you test for in your code.

Note that by default, once you enable tracing it will only apply to local requests. That prevents actual end users from seeing the tracing information. If you need to trace a web page from an offsite location, you should use a technique like the one shown previously (for query string activation). You'll also need to change some web.config settings to enable remote tracing.

6.7.2 Tracing Information

ASP.NET tracing automatically provides a lengthy set of standard, formatted information. Figure 6.15 shows what this information looks like. To build this example, you can start with any basic ASP.NET page. Shown here is a rudimentary ASP.NET page with just a label and a button.

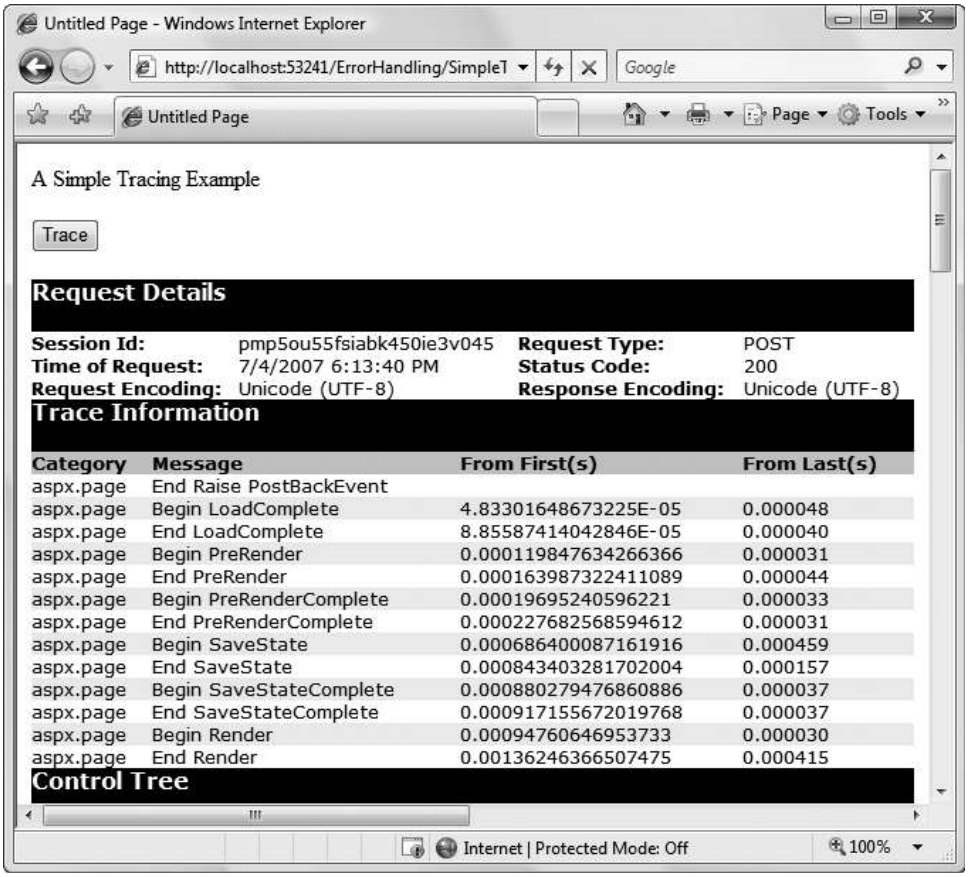
Figure 6.15: A Simple ASP.NET Page



Notes

On its own, this page does very little, displaying a single line of text. However, if you click the button, tracing is enabled by setting the Trace.IsEnabled property to true (as shown in the previous code snippet). When the page is rendered, it will include a significant amount of diagnostic information, as shown in Figure 6.16.

Figure 6.16: Tracing the Simple ASP.NET Page



Tracing information is provided in several different categories, which are described in the following sections. Depending on your page, you may not see all the tracing information.

Example: If the page request doesn't supply any query string parameters, you won't see the QueryString collection. Similarly, if there's no data being held in application or session state, you won't see those sections either.

Request Details

This section includes some basic information such as the current session ID, the time the web request was made, and the type of web request and encoding (Figure 6.17). Most of these details are fairly uninteresting, and you won't spend much time looking at them. The exception is the session ID it allows you to determine when a new session is created. (Sessions are used to store information for a specific user in between page requests.)

Notes

Figure 6.17: Request Details

Request Details

Session Id:	pmp5ou55fsiabk450ie3v045	**Request Type:**	POST
Time of Request:	7/4/2007 6:13:40 PM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Trace information shows the different stages of processing that the page went through before being sent to the client (Figure 6.18). Each section has additional information about how long it took to complete, as a measure from the start of the first stage (From First) and as a measure from the start of the previous stage (From Last). If you add your own trace messages (a technique described shortly), they will also appear in this section.

Figure 6.18: Trace Information

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	End Raise PostBackEvent		
aspx.page	Begin LoadComplete	4.83301648673225E-05	0.000048
aspx.page	End LoadComplete	8.85587414042846E-05	0.000040
aspx.page	Begin PreRender	0.000119847634266366	0.000031
aspx.page	End PreRender	0.000163987322411089	0.000044
aspx.page	Begin PreRenderComplete	0.00019695240596221	0.000033
aspx.page	End PreRenderComplete	0.000227682568594612	0.000031
aspx.page	Begin SaveState	0.000686400087161916	0.000459
aspx.page	End SaveState	0.000843403281702004	0.000157
aspx.page	Begin SaveStateComplete	0.000880279476860886	0.000037
aspx.page	End SaveStateComplete	0.000917155672019768	0.000037
aspx.page	Begin Render	0.00094760646953733	0.000030
aspx.page	End Render	0.00136246366507475	0.000415

Control Tree

The control tree shows you all the controls on the page, indented to show their hierarchy (which controls are contained inside other controls), as shown in Figure 6.19. In this simple page example, the control tree includes buttons named cmdWrite, cmdWrite_Category, cmdError, and cmdSession, all of which are explicitly defined in the web page markup. ASP.NET also adds literal controls automatically to represent spacing and any other static elements that aren't server controls (such as text or ordinary HTML tags). These controls appear in between the buttons in this example, and have automatically generated names like ctl00, ctl01, ctl02, and so on.

Figure 6.19: Control Tree

Control Tree

Control UniqueID	Type	Render Size Bytes (including children)	ViewState Size Bytes (excluding children)	ControlState Size Bytes (excluding children)
__Page	ASP.simpletrace_aspx	775	0	0
ctl02	System.Web.UI.LiteralControl	175	0	0
ctl00	System.Web.UI.HtmlControls.HtmlHead	46	0	0
ctl01	System.Web.UI.HtmlControls.HtmlTitle	33	0	0
ctl03	System.Web.UI.LiteralControl	14	0	0
form1	System.Web.UI.HtmlControls.HtmlForm	520	0	0
ctl04	System.Web.UI.LiteralControl	77	0	0
cmdTrace	System.Web.UI.WebControls.Button	67	0	0
ctl05	System.Web.UI.LiteralControl	12	0	0
ctl06	System.Web.UI.LiteralControl	20	0	0

Notes

One useful feature of this section is the Viewstate column, which tells you how many bytes of space are required to persist the current information in the control. This can help you gauge whether enabling control state is detracting from performance, particularly when working with data-bound controls such as the GridView.

Session State and Application State

These sections display every item that is in the current session or application state. Each item in the appropriate state collection is listed with its name, type, and value. If you're storing simple pieces of string information, the value is straightforward it's the actual text in the string. If you're storing an object, .NET calls the object's ToString() method to get an appropriate string representation. For complex objects that don't override ToString() to provide anything useful, the result may just be the class name.

Figure 6.20 shows the session state section after you've added two items to session state (an ordinary string and a DataSet object).

Figure 6.20: Session State

Session State

Session Key	Type	Value
TestString	System.String	This is just a string.
MyDataSet	System.Data.DataSet	System.Data.DataSet

Request Cookies and Response Cookies

These sections display the cookies that were sent by the web browser with the request for this page, and the cookies that were returned by the web server with the response. ASP.NET shows the content and the size of each cookie in bytes.

Figure 6.21 shows an example with a page that uses a cookie named Preferences that stores a single piece of information: a user name. In addition, the web browser receives a cookie named ASP.NET_ SessionId, which ASP.NET creates automatically to store the current session ID.

Figure 6.21: Cookies Collections

Request Cookies Collection

Name	Value	Size
ASP.NET_SessionId	lzydtbz0iw5dvoyhne2oevz3	42

Response Cookies Collection

Name	Value	Size
Preferences	(Name=Jackson Polenta)	32

There's one quirk with the list of cookies in the trace information. If you haven't created at least one custom cookie of your own, you won't see any cookies, including the ones that ASP.NET creates automatically (like the session cookie).

Notes

Case Study

Creative Web2Print Solution for an Online British Magazine

A creative adventure for the netizens, who get to explore their creative side, by striking a star appeal on their own photographs; a features-packed online creative canvass lets them design their custom magazine covers; finally they can seek to flaunt this stardom on social media or choose to buy designs online.

The Client

A fresh and happening online concept magazine from UK. Coverdude introduced a rage among Gen Y; a facility that lets them feed their appetite for earning a slice of celebrity glory by allowing them to create fake custom magazine covers, with their photograph. While the online users were looting fun, Coverdude chose to enhance the reach and utility of this application, by automating it with a slew of design presets, print options and very importantly, an opportunity to connect on social media.

The (Creative) Crux of the Matter

Having a nexus to various social media platforms was a clever thought up Coverdude´s sleeve. The advent of social media set a new precedence for the way ideas could be imagined on the internet. This radically changed the way the world used to connect. Social media has morphed people communications and suddenly the internet has become much more expressive. It´s a one-on-one, Word of Mouth phenomenon. The expression holds the key. It is this lucrative opportunity Coverdude wanted to leverage, of course, with a host of other creative attributes.

Coverdude asked Radixweb to prepare a platform from ground up, which forms into a creative addiction and an engaging tool for social media platforms. A web adventure where people can come to celebrate their individuality; by imagining a fun that reflects their aspirations; and then get to share this star appeal over social media platforms. Coverdude wished for an array of features, having immense scope for customization and creative liberty.

The requirements included:

1. Choice for the users to make their own magazine covers
2. Integration of magazine covers with social media platforms
3. Facility for buying/ordering the magazine covers online
4. Opportunity for site visitors to browse the creations designed by the users
5. Design presets/templates creation and control for the administrator
6. A feature-rich user-account having sections like Portfolio, Order History and the like
7. Opportunity for setting up Google Ads
8. SEO (Search Engine Optimization) friendly platform

Contd...

Notes

The Solution

Radixweb thoroughly analyzed the entire concept to design and develop the website keeping in mind the business needs as well as the creative gist of designing custom magazine covers for Coverdude.

The solution offered the following key features:

1. A vast canvass of design creativity using predefined fake magazine covers
2. A host of cover editing and customization features like shrinking, zooming, scaling, positioning and updating text using our flash designer tool
3. Integration of covers with social networking sites like Facebook, Yahoo, MySpace, Picaso, etc.
4. Opportunity to embed and share the cover link in social networking sites
5. A convenient account management system for users to manage the portfolio and order details
6. A section named ´Recent Creation´ lets users browse through the freshly uploaded content
7. PDF generation for viewing, downloading and printing
8. SEO-friendly website, in sync with the client´s web promotion goals over search engines like Google, Yahoo, etc.
9. Comprehensive Admin control facilitated with a Content Management System for managing design templates, orders, customers´ details and various types of content using the (CMS)
10. The administrator can also manage content likes title, keywords and metatags dynamically for the pages available in the front side

Coverdude got a slick and a simple to use customized design studio, which lets its online consumers make personalized fake magazine covers and order prints. A creative platform for one and all. A mature entertainment and networking resource with oodles of joy.

Everyday hundreds of users use Coverdude´s customized designer studio solution, to picture themselves in the aspirations they long for; someone turns a Super model, while someone might pose like a business tycoon; and finally they share their celebrity pomp with great excitement. On the other hand, for Coverdude, it´s indeed a very good commerce; offered to an ardent community of netizens as an easy-to-use fun and a good buy too.

6.8 Summary

- One of the most significant differences between an ordinary website and a professional web application is often in how it deals with errors.
- In this unit, you learned the different lines of defense you can use in .NET, including structured error handling, logging, custom error pages, and tracing.

6.9 Keywords

A Fatal Exception: A fatal exception (also called a critical or catastrophic error) is an event that cannot be properly handled to allow the application-or the operating system-to continue running.

Notes

Exception Handling: The process of trapping an exception and performing some sort of corrective procedure in response.

Exception: A signal that is generated when an unplanned or unexpected event occurs. Exceptions are typically caught by an exception handler and dealt with in an appropriate way.

Tracing: In ASP.NET, the process of capturing and displaying debugging information about a Web page as the page is running. Tracing information includes HTTP headers and control state.

6.10 Self Assessment

Fill in the blanks:

1. .NET searches for local error handlers and then checks for any active error handlers in the .....
2. ..... may still occur because of unforeseen circumstances, but they will be caught in the code and identified.
3. Every exception class derives from the base class .....
4. ..... provides a useful tool to browse through the exceptions in the .NET class library.
5. The ..... property is an extremely useful tool for component-based programming.
6. ..... exception handling is particularly flexible because it allows you to catch specific types of exceptions.

State whether the following statements are True or False:

7. Exception blocks work a little like conditional code.
8. When an exception is thrown, .NET do not tries to find a matching catch statement in the current method.
9. Custom exception classes should always inherit from System.ApplicationException, which itself derives from the base Exception class.
10. The .NET Framework provides a wide range of logging tools.
11. ASP.NET do not allows you to replace the generic error page with a custom error page of your own devising.
12. There are several ways to switch on tracing.

6.11 Review Questions

1. How do you turn on tracing for a specific page?
2. What are the differences between the Trace.Write and Trace.Warn methods?
3. What are the three possible arguments to the Trace.Write and Trace.Warn methods?
4. How do you set a breakpoint in your code?
5. How do you determine the current value of a variable when the application is stopped at a breakpoint?
6. How do you modify the value of a variable while the application is running?
7. What information can you find in the Locals window?

Notes

8. What are the differences between syntax and logic errors?
9. What setting do you need to specify before you can use a custom error page?
10. How do you specify that a specific page should use its own custom error page instead of the application-wide pages?

Answers: Self Assessment

1.	calling code	2.	Errors
3.	System.Exception	4.	Visual Studio
5.	InnerException	6.	Structured
7.	True	8.	False
9.	True	10.	True
11.	False	12.	True

6.12 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Notes

Unit 7: Validation

CONTENTS

Objectives

Introduction

7.1 Understanding Validation

7.1.1 Server-side Validation

7.1.2 Client-side Validation

7.2 The Validation Controls

7.2.1 A Simple Validation Example

7.2.2 Other Display Options

7.2.3 Manual Validation

7.2.4 Validating with Regular Expressions

7.3 Validation Groups

7.4 Summary

7.5 Keywords

7.6 Self Assessment

7.7 Review Questions

7.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe validation
- Explain validation controls

Introduction

This unit takes a look at a specific type of server control you find in the Toolbox window: The validation server control.

Validation server controls are a series of controls that enable you to work with the information your end users input into the form elements of the applications you build. These controls work to ensure the validity of the data being placed in the form.

In this unit, you'll learn how to use the validation controls in an ASP.NET web page, and how to get the most out of them with sophisticated regular expressions, custom validation functions, and more. And as usual, you'll peer under the hood to see how ASP.NET implements these features.

Notes

7.1 Understanding Validation

As a seasoned developer, you probably realize users will make mistakes. What's particularly daunting is the range of possible mistakes that users can make. Here are some common examples:

1. Users might ignore an important field and leave it blank.
2. Users might try to type a short string of nonsense to circumvent a required field check, thereby creating endless headaches on your end. For example, you might get stuck with an invalid e-mail address that causes problems for your automatic e-mailing program.
3. Users might make an honest mistake, such as entering a typing error, entering a nonnumeric character in a number field, or submitting the wrong type of information. They might even enter several pieces of information that are individually correct but when taken together are inconsistent (for example, entering a MasterCard number after choosing Visa as the payment type).
4. Malicious users might try to exploit a weakness in your code by entering carefully structured wrong values. For example, they might attempt to cause a specific error that will reveal sensitive information. A more dramatic example of this technique is the SQL injection attack, where user-supplied values change the operation of a dynamically constructed database command.

A web application is particularly susceptible to these problems, because it relies on basic HTML input controls that don't have all the features of their Windows counterparts. For example, a common technique in a Windows application is to handle the KeyPress event of a text box, check to see whether the current character is valid, and prevent it from appearing if it isn't. This technique makes it easy to create a text box that accepts only numeric input.

In web applications, however, you don't have that sort of fine-grained control. To handle a KeyPress event, the page would have to be posted back to the server every time the user types a letter, which would slow down the application hopelessly. Instead, you need to perform all your validation at once when a page (which may contain multiple input controls) is submitted. You then need to create the appropriate user interface to report the mistakes. Some websites report only the first incorrect field, while others use a table, list, or window to describe them all. By the time you've perfected your validation strategy, you'll have spent a considerable amount of effort writing tedious code.

ASP.NET aims to save you this trouble and provide you with a reusable framework of validation controls that manages validation details by checking fields and reporting on errors automatically. These controls can even use client-side JavaScript to provide a more dynamic and responsive interface while still providing ordinary validation for older browsers (often referred to as down-level browsers).

Table 7.1: Validator Controls

Control Class	Description
RequiredFieldValidator	Validation succeeds as long as the input control doesn't contain an empty string.
RangeValidator	Validation succeeds if the input control contains a value within a specific numeric, alphabetic, or date range.
CompareValidator	Validation succeeds if the input control contains a value that matches the value in another input control, or a fixed value that you specify.
RegularExpressionValidator	Validation succeeds if the value in an input control matches a specified regular expression.
CustomValidator	Validation is performed by a user-defined function.

Notes

ASP.NET provides five validator controls, which are described in Table 7.1. Four are targeted at specific types of validation, while the fifth allows you to apply custom validation routines. You'll also see a ValidationSummary control in the Toolbox, which gives you another option for showing a list of validation error messages in one place.

Each validation control can be bound to a single input control. In addition, you can apply more than one validation control to the same input control to provide multiple types of validation.

If you use the RangeValidator, CompareValidator, or RegularExpressionValidator, validation will automatically succeed if the input control is empty, because there is no value to validate. If this isn't the behavior you want, you should also add a RequiredFieldValidator and link it to the same input control. This ensures that two types of validation will be performed, effectively restricting blank values.

7.1.1 Server-side Validation

You can use the validator controls to verify a page automatically when the user submits it or manually in your code. The first approach is the most common.

When using automatic validation, the user receives a normal page and begins to fill in the input controls. When finished, the user clicks a button to submit the page. Every button has a CausesValidation property, which can be set to true or false. What happens when the user clicks the button depends on the value of the CausesValidation property:

1. If CausesValidation is false, ASP.NET will ignore the validation controls, the page will be posted back, and your event-handling code will run normally.
2. If CausesValidation is true (the default), ASP.NET will automatically validate the page when the user clicks the button. It does this by performing the validation for each control on the page. If any control fails to validate, ASP.NET will return the page with some error information, depending on your settings. Your click event-handling code may or may not be executed—meaning you'll have to specifically check in the event handler whether the page is valid.

Based on this description, you'll realize that validation happens automatically when certain buttons are clicked. It doesn't happen when the page is posted back because of a change event (such as choosing a new value in an AutoPostBack list) or if the user clicks a button that has CausesValidation set to false. However, you can still validate one or more controls manually and then make a decision in your code based on the results.

Notes Many other button-like controls that can be used to submit the page also provide the CausesValidation property. Examples include the LinkButton, ImageButton, and BulletedList.

7.1.2 Client-side Validation

In most modern browsers (including Internet Explorer 5 or later and any version of Firefox), ASP.NET automatically adds JavaScript code for client-side validation. In this case, when the user clicks a CausesValidation button, the same error messages will appear without the page needing to be submitted and returned from the server. This increases the responsiveness of your web page.

Notes

However, even if the page validates successfully on the client-side, ASP.NET still revalidates it when it's received at the server. This is because it's easy for an experienced user to circumvent client-side validation.

Example: A malicious user might delete the block of JavaScript validation code and continue working with the page. By performing the validation at both ends, ASP.NET makes sure your application can be as responsive as possible while also remaining secure.

Task Write a sample code fragment that will check whether the first character of the input is upper or lowercase.

7.2 The Validation Controls

The validation controls are found in the System.Web.UI.WebControls namespace and inherit from the BaseValidator class. This class defines the basic functionality for a validation control. Table 7.2 describes its key properties.

Table 7.2: Properties of the BaseValidator Class

Property	Description
Control To Validate	Identifies the control that this validator will check. Each validator can verify the value in one input control. However, it's perfectly reasonable to "stack" validators in other words, attach several validators to one input control to perform more than one type of error checking.
ErrorMessage and ForeColor	If validation fails, the validator control can display a text message (set by the ErrorMessage property). By changing the ForeColor, you can make this message stand out in angry red lettering.
Display	Allows you to configure whether this error message will be inserted into the page dynamically when it's needed (Dynamic) or whether an appropriate space will be reserved for the message (Static). Dynamic is useful when you're placing several validators next to each other. That way, the space will expand to fit the currently active error indicators, and you won't be left with any unseemly whitespace. Static is useful when the validator is in a table and you don't want the width of the cell to collapse when no message is displayed Finally, you can also choose None to hide the error message altogether.
IsValid	After validation is performed, this returns true or false depending on whether it succeeded or failed. Generally, you'll check the state of the entire page by looking at its IsValid property instead to find out if all the validation controls succeeded.
Enabled	When set to false, automatic validation will not be performed for this control when the page is submitted.
EnableClientScript	If set to true, ASP.NET will add JavaScript and DHTML code to allow client-side validation on browsers that support it.

When using a validation control, the only properties you need to implement are ControlToValidate and ErrorMessage. In addition, you may need to implement the properties that are used for your specific validator. Table 7.3 outlines these properties.

Notes

Table 7.3: Validator-Specific Properties

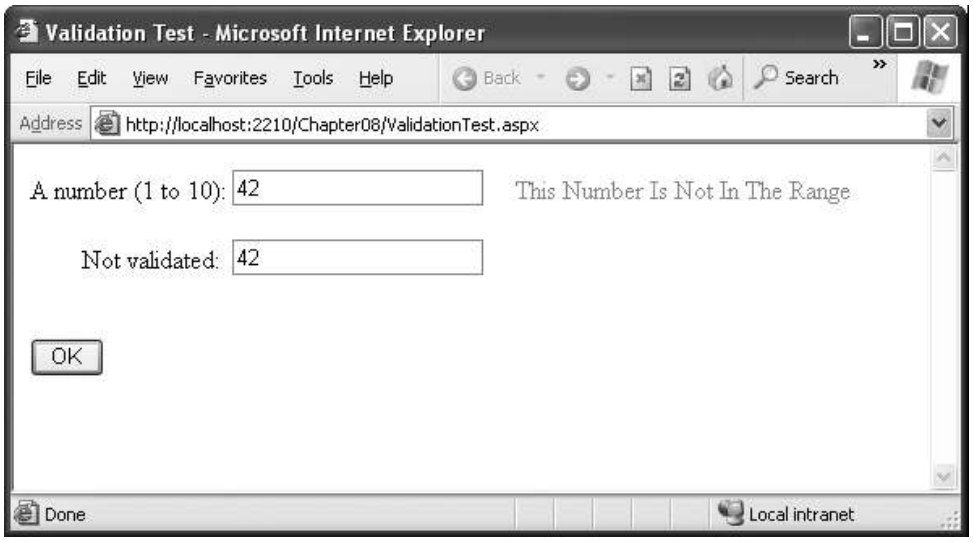
Validator Control	Added Members
RequiredFieldValidator	None required
RangeValidator	MaximumValue, MinimumValue, Type
CompareValidator	ControlToCompare, Operator, Type, ValueToCompare
RegularExpressionValidator	ValidationExpression
CustomValidator	ClientValidationFunction, ValidateEmptyText, ServerValidate event

7.2.1 A Simple Validation Example

To understand how validation works, you can create a simple web page. This test uses a single Button web control, two TextBox controls, and a RangeValidator control that validates the first text box. If validation fails, the RangeValidator control displays an error message, so you should place this control immediately next to the TextBox it's validating. The second text box does not use any validation.

Figure 7.1 shows the appearance of the page after a failed validation attempt.

Figure 7.1: Failed Validation



In addition, place a Label control at the bottom of the form. This label will report when the page has been posted back and the event-handling code has executed. Disable its EnableViewState property to ensure that it will be cleared every time the page is posted back.

The markup for this page defines a RangeValidator control, sets the error message, identifies the control that will be validated, and requires an integer from 1 to 10. These properties are set in the .aspx file, but they could also be configured in the event handler for the Page.Load event. The Button automatically has its CauseValidation property set to true, because this is the default.

```
A number (1 to 10):
<asp:TextBox id="txtValidated" runat="server" />
<asp:RangeValidator id="RangeValidator" runat="server"
ErrorMessage="This Number Is Not In The Range"
```

Notes

```
ControlToValidate="txtValidated"
MaximumValue="10" MinimumValue="1"
Type="Integer" />
<br /><br />
Not validated:
<asp:TextBox id="txtNotValidated" runat="server" /><br /><br />
<asp:Button id="cmdOK" runat="server" Text="OK" OnClick="cmdOK_Click" />
<br /><br />
<asp:Label id="lblMessage" runat="server"
EnableViewState="False" />
Finally, here is the code that responds to the button click:
protected void cmdOK_Click(Object sender, EventArgs e)
{
lblMessage.Text = "cmdOK_Click event handler executed.";
}
```

If you're testing this web page in a modern browser, you'll notice an interesting trick. When you first open the page, the error message is hidden. But if you type an invalid number (remember, validation will succeed for an empty value) and press the Tab key to move to the second text box, an error message will appear automatically next to the offending control. This is because ASP.NET adds a special JavaScript function that detects when the focus changes. The actual implementation of this JavaScript code is somewhat complicated, but ASP.NET handles all the details for you automatically. As a result, if you try to click the OK button with an invalid value in txtValidated, your actions will be ignored and the page won't be posted back.

Not all browsers will support client-side validation. To see what will happen on a downlevel browser, set the RangeValidator.EnableClientScript property to false, and rerun the page. Now error messages won't appear dynamically as you change focus. However, when you click the OK button, the page will be returned from the server with the appropriate error message displayed next to the invalid control.

The potential problem in this scenario is that the click event-handling code will still execute, even though the page is invalid. To correct this problem and ensure that your page behaves the same on modern and older browsers, you must specifically abort the event code if validation hasn't been performed successfully.

```
protected void cmdOK_Click(Object sender, EventArgs e)
{
// Abort the event if the control isn't valid.
if (!RangeValidator.IsValid) return;
lblMessage.Text = "cmdOK_Click event handler executed.";
}
```

This code solves the current problem, but it isn't much help if the page contains multiple validation controls. Fortunately, every web form provides its own IsValid property. This property will be false if any validation control has failed. It will be true if all the validation controls completed successfully. If validation was not performed (for example, if the validation controls are disabled or if the button has CausesValidation set to false), you'll get an HttpException when you attempt to read the IsValid property.

```
protected void cmdOK_Click(Object sender, EventArgs e)
{
// Abort the event if any control on the page is invalid.
```

Notes

```
if (!Page.IsValid) return;
lblMessage.Text = "cmdOK_Click event handler executed.";
}
```

Notes Remember, client-side validation is just nice frosting on top of your application. Server-side validation will always be performed, ensuring that crafty users can't "spoof" pages.

7.2.2 Other Display Options

In some cases, you might have already created a carefully designed form that combines multiple input fields. Perhaps you want to add validation to this page, but you can't reformat the layout to accommodate all the error messages for all the validation controls. In this case, you can save some work by using the ValidationSummary control.

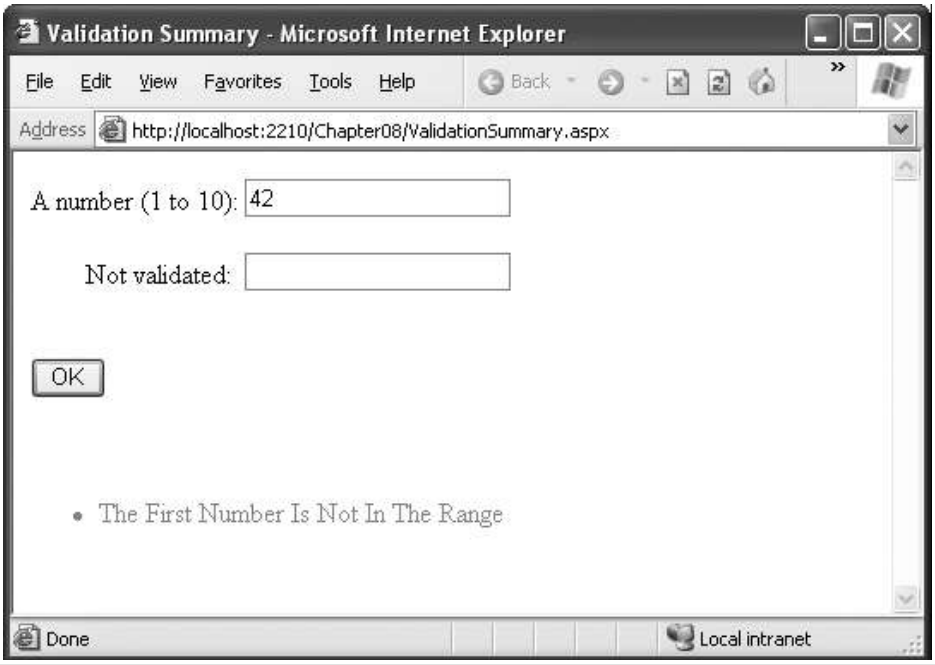
To try this, set the Display property of the RangeValidator control to None. This ensures the error message will never be displayed. However, validation will still be performed and the user will still be prevented from successfully clicking the OK button if some invalid information exists on the page.

Next, add the ValidationSummary in a suitable location (such as the bottom of the page):

```
<asp:ValidationSummary id="Errors" runat="server" />
```

When you run the page, you won't see any dynamic messages as you enter invalid information and tab to a new field. However, when you click the OK button, the ValidationSummary will appear with a list of all error messages, as shown in Figure 7.2. In this case, it retrieves one error message (from the RangeValidator control). However, if you had a dozen validators, it would retrieve all their error messages and create a list.

Figure 7.2: The Validation Summary



Notes

When the ValidationSummary displays the list of errors, it automatically retrieves the value of the ErrorMessage property from each validator. In some cases, you'll want to display a full message in the summary and some sort of visual indicator next to the offending control.

Example: Many websites use an error icon or an asterisk to highlight text boxes with invalid input. You can use this technique with the help of the Text property of the validators. Ordinarily, Text is left empty. However, if you set both Text and ErrorMessage, the ErrorMessage value will be used for the summary while the Text value is displayed in the validator. (Of course, you'll need to make sure you aren't also setting the Display property of your validator to None, which hides it completely.)

Here's an example of a validator that includes a detailed error message (which will appear in the Validation Summary) and an asterisk indicator (which will appear in the validator, next to the control that has the problem):

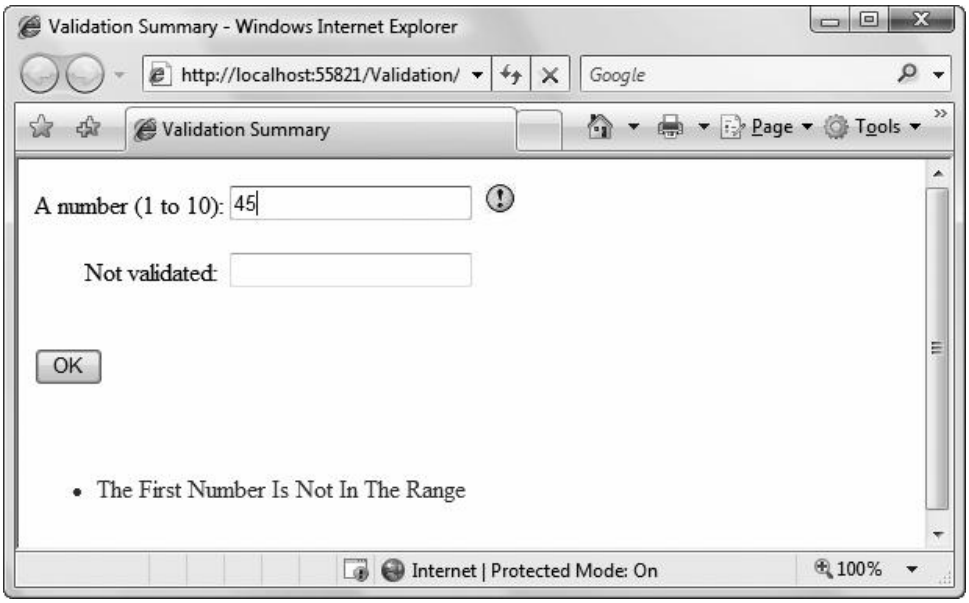
```
<asp:RangeValidator id="RangeValidator" runat="server"
Text="*" ErrorMessage="The First Number Is Not In The Range"
ControlToValidate="txtValidated"
MaximumValue="10" MinimumValue="1" Type="Integer" />
```

You can even get a bit fancier by replacing the plain asterisk with a snippet of more interesting HTML. Here's an example that uses the <img> tag to add a small error icon image when validation fails:

```
<asp:RangeValidator id="RangeValidator" runat="server"
Text="<img src='ErrorIcon.gif' />" alt='Error' ... />
```

Figure 7.3 shows this validator in action.

Figure 7.3: A Validation Summary and an Error Indicator



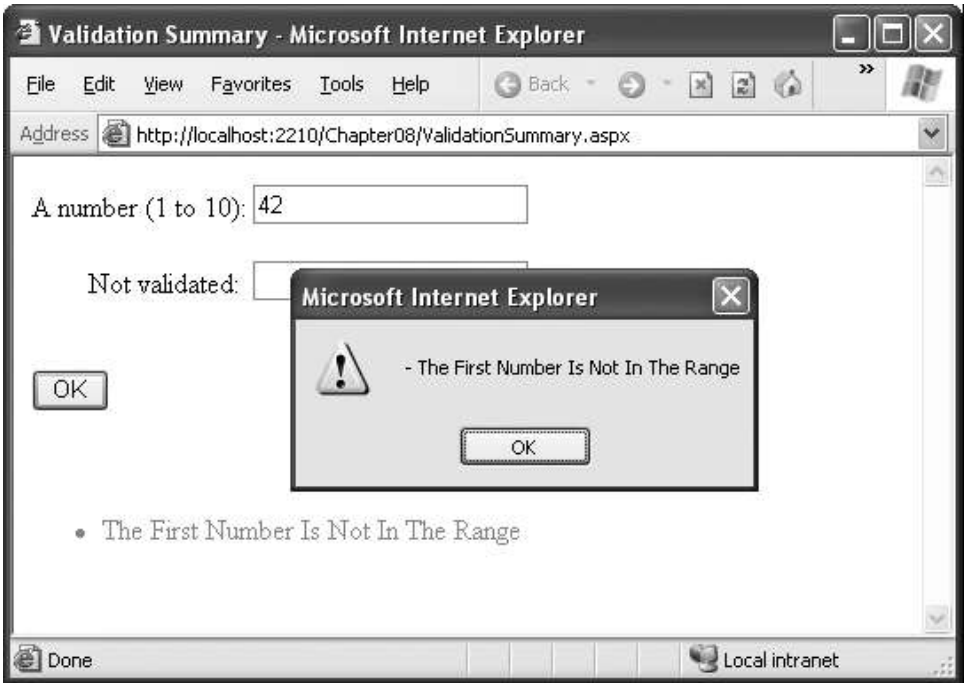
The ValidationSummary control provides some useful properties you can use to fine-tune the error display. You can set the HeaderText property to display a special title at the top of the list (such as Your page contains the following errors:). You can also change the ForeColor and choose a DisplayMode. The possible modes are BulletList (the default), List, and SingleParagraph.

Notes

Finally, you can choose to have the validation summary displayed in a pop-up dialog box instead of on the page (Figure 7.3). This approach has the advantage of leaving the user interface of the page untouched, but it also forces the user to dismiss the error messages by closing the window before being able to modify the input controls. If users will need to refer to these messages while they fix the page, the inline display is better.

To show the summary in a dialog box, set the ShowMessageBox property of the ValidationSummary to true. Keep in mind that unless you set the ShowSummary property to false, you'll see both the message box and the in-page summary (Figure 7.4).

Figure 7.4: A Message Box Summary



7.2.3 Manual Validation

Your final option is to disable validation and perform the work on your own, with the help of the validation controls. This allows you to take other information into consideration or create a specialized error message that involves other controls (such as images or buttons).

You can create manual validation in one of three ways:

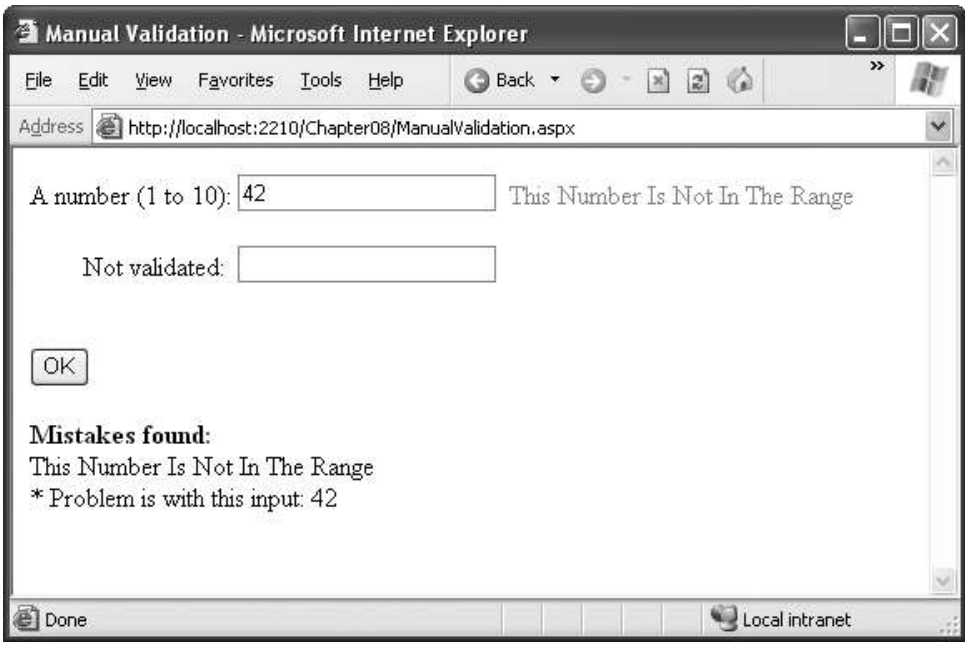
1. Use your own code to verify values. In this case, you won't use any of the ASP.NET validation controls.
2. Disable the EnableClientScript property for each validation control. This allows an invalid page to be submitted, after which you can decide what to do with it depending on the problems that may exist.
3. Add a button with CausesValidation set to false. When this button is clicked, manually validate the page by calling the Page.Validate() method. Then examine the IsValid property, and decide what to do.

The next example uses the second approach. Once the page is submitted, it examines all the validation controls on the page by looping through the Page.Validators collection. Every time

Notes

it finds a control that hasn't validated successfully, it retrieves the invalid value from the input control and adds it to a string. At the end of this routine, it displays a message that describes which values were incorrect, as shown in Figure 7.5.

Figure 7.5: Manual Validation



This technique adds a feature that wouldn't be available with automatic validation, which uses the ErrorMessage property. In that case, it isn't possible to include the actual incorrect values in the message.

Here's the event handler that checks for invalid values:

```
protected void cmdOK_Click(Object sender, EventArgs e)
{
string errorMessage = "<b>Mistakes found:</b><br />";
// Search through the validation controls.
foreach (BaseValidator ctrl in this.Validators)
{
if (!ctrl.IsValid)
{
errorMessage += ctrl.ErrorMessage + "<br />";
// Find the corresponding input control, and change the
// generic Control variable into a TextBox variable.
// This allows access to the Text property.
TextBox ctrlInput =
(TextBox)this.FindControl(ctrl.ControlToValidate);
errorMessage += " * Problem is with this input: ";
errorMessage += ctrlInput.Text + "<br />";
}
}
```

Notes

```
lblMessage.Text = errorMessage;
}
```

This example uses an advanced technique: the Page.FindControl() method. It's required because the ControlToValidate property of each validator simply provides a string with the name of a control, not a reference to the actual control object. To find the control that matches this name (and retrieve its Text property), you need to use the FindControl() method. Once the code has retrieved the matching text box, it can perform other tasks such as clearing the current value, tweaking a property, or even changing the text box color. Note that the FindControl() method returns a generic Control reference, because you might search any type of control. To access all the properties of your control, you need to cast it to the appropriate type.

Task Write a code fragment that will return the three primary section of a time variable.

7.2.4 Validating with Regular Expressions

One of ASP.NET's most powerful validation controls is the RegularExpressionValidator, which validates text by determining whether or not it matches a specific pattern.

Example: e-mail addresses, phone numbers, and file names are all examples of text that has specific constraints. A phone number must be a set number of digits, an e-mail address must include exactly one @ character (with text on either side), and a file name can't include certain special characters like \ and ?. One way to define patterns like these is with regular expressions.

Regular expressions have appeared in countless other languages and gained popularity as an extremely powerful way to work with strings. In fact, Visual Studio even allows programmers to perform a search-and-replace operation in their code using a regular expression (which may represent a new height of computer geekdom). Regular expressions can almost be considered an entire language of their own.

Literals and Metacharacters

All regular expressions consist of two kinds of characters: literals and metacharacters. Literals are not unlike the string literals you type in code. They represent a specific defined character. For example, if you search for the string literal "l", you'll find the character l and nothing else.

Metacharacters provide the true secret to unlocking the full power of regular expressions. You're probably already familiar with two metacharacters from the DOS world (? and *). Consider the command-line expression shown here:

```
Del *.*
```

The expression *.* contains one literal (the period) and two metacharacters (the asterisks). This translates as "delete every file that starts with any number of characters and ends with an extension of any number of characters (or has no extension at all)." Because all files in DOS implicitly have extensions, this has the well-documented effect of deleting everything in the current directory.

Another DOS metacharacter is the question mark, which means "any single character." For example, the following statement deletes any file named hello that has an extension of exactly one character.

```
Del hello.?
```

Notes

The regular expression language provides many flexible metacharacters—far more than the DOS command line. For example, \s represents any whitespace character (such as a space or tab). \d represents any digit. Thus, the following expression would match any string that started with the numbers 333, followed by a single whitespace character and any three numbers. Valid matches would include 333 333 and 333 945 but not 334 333 or 3334 945.

```
333\s\d\d\d
```

One aspect that can make regular expressions less readable is that they use special metacharacters that are more than one character long. In the previous example, \s represents a single character, as does \d, even though they both occupy two characters in the expression.

You can use the plus (+) sign to represent a repeated character. For example, 5+7 means "one or more occurrences of the character 5, followed by a single 7." The number 57 would match, as would 555557. You can also use parentheses to group a subexpression. For example, (52)+7 would match any string that started with a sequence of 52. Matches would include 527, 52527, 5252527, and so on.

You can also delimit a range of characters using square brackets. [a-f] would match any single character from a to f (lowercase only). The following expression would match any word that starts with a letter from a to f, contains one or more "word" characters (letters), and ends with ing—possible matches include acting and developing.

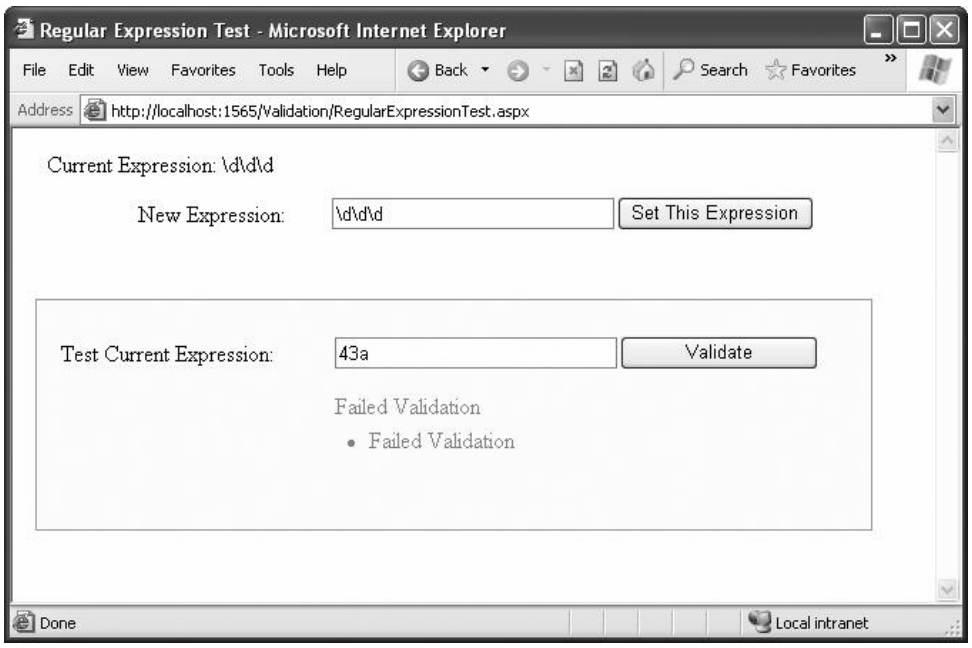
```
[a-f]\w+ing
```

The following is a more useful regular expression that can match any e-mail address by verifying that it contains the @ symbol. The dot is a metacharacter used to indicate any character except newline. However, some invalid e-mail addresses would still be allowed, including those that contain spaces and those that don't include a dot (.). You'll see a better example a little later in the customer form example.

```
.+@.+
```

Finding a Regular Expression

Figure 7.6: A Regular Expression Test Page



Notes

Clearly, picking the perfect regular expression may require some testing. In fact, numerous reference materials (on the Internet and in paper form) include useful regular expressions for validating common values such as postal codes. To experiment, you can use the simple RegularExpressionTest page included with the online samples, which is shown in Figure 7.6. It allows you to set a regular expression that will be used to validate a control. Then you can type in some sample values and see whether the regular expression validator succeeds or fails.

The code is quite simple. The Set This Expression button assigns a new regular expression to the RegularExpressionValidator control (using whatever text you have typed). The Validate button simply triggers a postback, which causes ASP.NET to perform validation automatically. If an error message appears, validation has failed. Otherwise, it's successful.

```
public partial class RegularExpressionTest : System.Web.UI.Page
{
protected void cmdSetExpression_Click(Object sender, EventArgs e)
{
TestValidator.ValidationExpression = txtExpression.Text;
lblExpression.Text = "Current Expression: ";
lblExpression.Text += txtExpression.Text;
}
}
```

Table 7.4 shows some of the fundamental regular expression building blocks. If you need to match a literal character with the same name as a special character, you generally precede it with a \ character. For example, *hello* matches *hello* in a string, because the special asterisk (*) character is preceded by a slash (\).

Table 7.4: Regular Expression Characters

Character	Description
*	Zero or more occurrences of the previous character or subexpression. For example, 7*8 matches 7778 or just 8.
+	One or more occurrences of the previous character or subexpression. For example, 7+8 matches 7778 but not 8.
()	Groups a subexpression that will be treated as a single element. For example, (78)+ matches 78 and 787878.
{}	The previous character (or subexpression) can occur from m to n times. For example, A{1,3} matches A, AA, or AAA.
|	Either of two matches. For example, 8|6 matches 8 or 6.
[]	Matches one character in a range of valid characters. For example, [A-C] matches A, B, or C.
[^]	Matches a character that isn't in the given range. For example, [^A-B] matches any character except A and B.
.	Any character except newline. For example, .here matches where and there.
\s	Any whitespace character (such as a tab or space).
\S	Any nonwhitespace character.
\d	Any digit character.
\D	Any character that isn't a digit.
\w	Any "word" character (letter, number, or underscore).
\W	Any character that isn't a "word" character (letter, number, or underscore).

Notes

Table 7.5 shows a few common (and useful) regular expressions.

Table 7.5: Commonly used Regular Expressions

Content	Regular Expression	Description
E-mail address*	\S+@\S+\.\S+	Check for an at (@) sign and dot (.) and allow nonwhitespace characters only.
Password	\w+	Any sequence of one or more word characters (letter, space, or underscore).
Specific-length password	\w{4,10}	A password that must be at least four characters long but no longer than ten characters.
Advanced password	[a-zA-Z]\w{3,9}	As with the specific-length password, this regular expression will allow four to ten total characters. The twist is that the first character must fall in the range of a–z or A–Z (that is to say. it must start with a nonaccented ordinary letter).
Another advanced password	[a-zA-Z]\w*\d+\w*	This password starts with a letter character, followed by zero or more word characters, one or more digits, and then zero or more word characters. In short, it forces a password to contain one or more numbers somewhere inside it. You could use a similar pattern to require two numbers or any other special character.
Limited-length field	\S{4,10}	Like the password example, this allows four to ten characters, but it allows special characters (asterisks, ampersands, and so on).
U.S. Social Security number	\d{3}-\d{2}-\d{4}	A sequence of three, two, then four digits, with each group separated by a dash. You could use a similar pattern when requiring a phone number.

Some logic is much more difficult to model in a regular expression. An example is the Luhn algorithm, which verifies credit card numbers by first doubling every second digit, then adding these doubled digits together, and finally dividing the sum by ten. The number is valid (although not necessarily connected to a real account) if there is no remainder after dividing the sum. To use the Luhn algorithm, you need a CustomValidator control that runs this logic on the supplied value. (You can find a detailed description of the Luhn algorithm at http:// en.wikipedia.org/ wiki/Luhn_formula.)

7.3 Validation Groups

In more complex pages, you might have several distinct groups of controls, possibly in separate panels. In these situations, you may want to perform validation separately. For example, you might create a form that includes a box with login controls and a box underneath it with the controls for registering a new user. Each box includes its own submit button, and depending on which button is clicked, you want to perform the validation just for that section of the page.

This scenario is possible thanks to a feature called validation groups. To create a validation group, you need to put the input controls, the validators, and the CausesValidation button controls into the same logical group. You do this by setting the ValidationGroup property of every control with the same descriptive string (such as "LoginGroup" or "NewUserGroup"). Every control that provides a CausesValidation property also includes the ValidationGroup property.

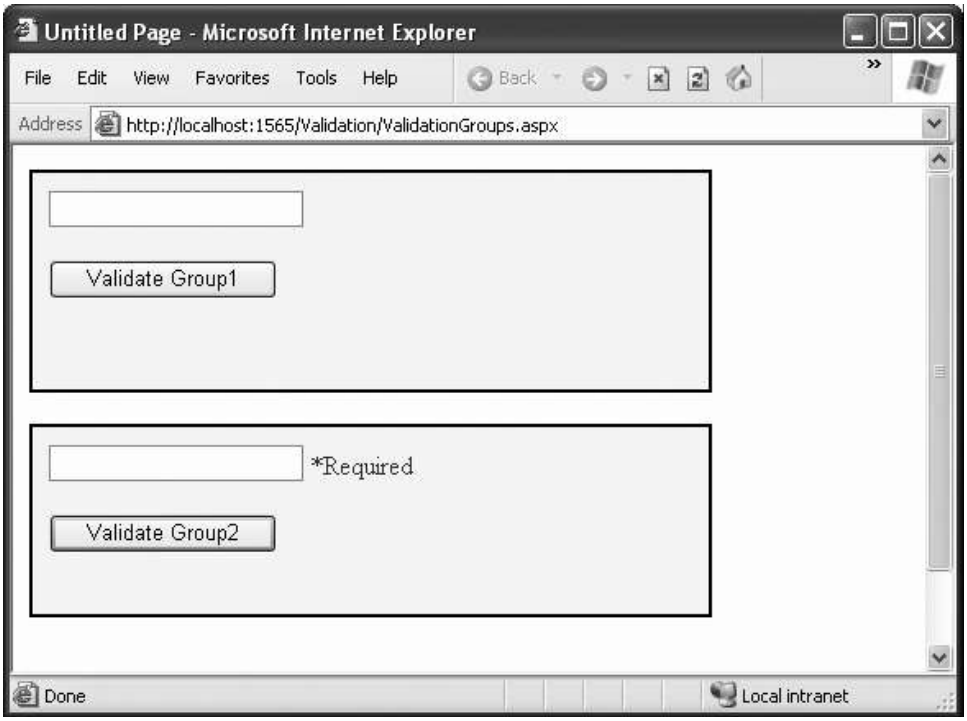
Example: The following page defines two validation groups, named Group1 and Group2. The controls for each group are placed into separate Panel controls.

Notes

```
<form id="form1" runat="server">
<asp:Panel ID="Panel1" runat="server">
<asp:TextBox ID="TextBox1" ValidationGroup="Group1" runat="server" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
ErrorMessage="*Required" ValidationGroup="Group1"
runat="server" ControlToValidate="TextBox1" />
<asp:Button ID="Button1" Text="Validate Group1"
ValidationGroup="Group1" runat="server" />
</asp:Panel> <br />
<asp:Panel ID="Panel2" runat="server">
<asp:TextBox ID="TextBox2" ValidationGroup="Group2"
runat="server" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator2"
ErrorMessage="*Required" ValidationGroup="Group2"
ControlToValidate="TextBox2" runat="server" />
<asp:Button ID="Button2" Text="Validate Group2"
ValidationGroup="Group2" runat="server" />
</asp:Panel>
</form>
```

If you click the button in the topmost Panel, only the first text box is validated. If you click the button in the second Panel, only the second text box is validated (Figure 7.7).

Figure 7.7: Grouping Controls for Validation



What happens if you add a new button that doesn't specify any validation group? In this case, the button validates every control that isn't explicitly assigned to a named validation group. In the

Notes

current example, no controls fit the requirement, so the page is posted back successfully and deemed to be valid.

If you want to make sure a control is always validated, regardless of the validation group of the button that's clicked, you'll need to create multiple validators for the control, one for each group (and one with no validation group).

Task

Write a code fragment that will check to see whether the single selections of two list boxes are the same.

Case Study

Slicing up the Cloud

When Simon Ellis looked at cloud-based computing, he saw one thing: opportunity. The veteran software architect and engineer envisioned an innovative way for IT professionals and lab managers to boost their productivity and performance while reducing overhead. By replacing physical systems with virtual ones, enterprises would be able to defer or eliminate costs associated with in-house labs, departmental testing infrastructures, and training environments. Further, his solution would be ideal for use by sales engineers, allowing them to swiftly, easily, and securely deliver application demos, evaluations, and Proofs of Concept (POCs).

As Ellis' vision began to take shape, he realized that others were likely seeing and acting on the same potential he'd observed. Time was of the essence – if his idea were to be successful, he would have to move quickly to turn his forward-looking concept into gainful reality. Ellis and the team at his newly founded startup, LabSlice, began immediately seeking out tools and technologies that would enable them to rapidly develop and deliver their product to market.

"As one startup among a multitude of other startups trying to capitalize on recent technology opportunities like virtualized lab environments, it is critical that you get your product to market as soon as possible," says Ellis. "Our main goal at LabSlice was to quickly get the product released, as our target market was starting to get crowded with competing products."

A Timely Solution: Telerik RadControls

The LabSlice application is a Virtual Lab Management solution that capitalizes on the benefits of cloud computing, such as improved cost-avoidance, lowered risk, and elevated productivity. The application enables replacement of physical systems incurring maintenance, licensing, and operational expenses with machines pulled directly from the Amazon Elastic Compute Cloud (Amazon EC2) operated by Amazon Web Services? (AWS). With LabSlice, enterprises can create a multi-user, policy-controlled self-service lab environment using virtual machines. These machines can be easily and securely shared internally with peers or externally with customers and business partners via email. It is a sophisticated, next-generation solution.

The team quickly realized that selection of a strong UI toolkit would be paramount to the application's success. It needed a robust toolset that would both fit seamlessly into the team's ASP.NET AJAX development environment and allow it to meet an aggressive

Contd...

Notes

six-month production schedule. Ellis turned to his team of developers for assistance, scouring numerous online forums, websites, and blogs, in an effort to determine what technologies being used successfully in existing projects. After receiving repeated recommendations from other developers, Ellis and his team settled on Telerik's RadControls for ASP.NET AJAX. The integrated controls suite offered the features, functionality, and support the team needed, and was cited by other developers for its ease-of-use and swift assimilation into project ecosystems. It proved to be the timely solution the team needed to jumpstart the application's development.

"We needed strong controls for grid manipulation, ajaxification of the interface, and Windows management. The market is full of these types of controls, but just a quick review will show that many are buggy, disjointed, and lack the strong support we needed," he recalls. "Getting started with Telerik controls was a trivial affair; within a week we were able to demonstrate our solution fully integrated with the new controls."

LabSlice Comes to Life

With RadControls for ASP.NET AJAX in place, Ellis' team was ready to bring LabSlice.com to life. The toolkit enabled developers to build a sleek customer-facing web interface featuring consistent UI elements backed by rich functionality. The suite's out-of-the-box operation and wide variety of pre-made themes spared the team from spending time on extensive customizations and typical challenges of browser-based UI programming, allowing a greater focus on development of key product features.

Deploying an array of RadControls components ranging from RadGrid and RadRotator to RadWindow and RadAjax, the LabSlice team was surprised at the swift pace of development, making significant progress within just one week. Telerik's RadGrid controls were used extensively throughout the application for generating, retrieving, and displaying list data, such as available machines in a given lab, active lab users, and recent log activity. RadGrid's exceptional flexibility offered the LabSlice team effortless look-and-feel customizability, robust management of large data volumes like activity logs, and allowed them to interact with the client via easy-to-use events built right into the control.

"Because Telerik provided a lot of the RadGrid events we needed right out of the box, we were able to easily achieve the user interactivity we required, without having to play around with messy-looking JavaScript code," notes Ellis.

The LabSlice application relies on efficient and effective workflow management, a function facilitated by custom wizard controls created around the RadWindow control. Pop-up RadWindow modal boxes are used to display relevant combo boxes, tree controls, and text-based information relevant to workflows. In addition, the control was implemented in user management features, including password changes and recovery.

RadToolBar was used at the top of most LabSlice controls, to present options that can be used with the client. With RadToolBar, the team could show, hide, or disable button based on user properties, or provide simple-to-use drop-down menus in areas with large numbers of available options. The technology's ability to deliver clean interfaces and neat encapsulation rescued developers from a potential coding time-sink.

"We wanted to use recurring patterns of the RadGrid, RadWindow and RadToolBar throughout the application to provide a consistent look-and-feel, and capitalize on as much of RadControls' built-in behaviors as we possibly could," says Ellis. "By doing this we were able to avoid much of the custom coding normally associated with a web app, and the troubles often encountered when trying to make custom code work across multiple platforms and browsers. "

Contd...

Notes

Although the LabSlice team encountered few problems during development, they were easily able overcome these challenges with the help of Telerik's industry-leading support program and extensive resource library.

"Any issues we had were referred to the Telerik staff, who responded promptly and helped us continue on with our development tasks," states Ellis. "But by far, our best resource was the online demos, which provided sufficient depth and guidance to help us with the use of all the controls. Other outstanding concerns were easily fixed by referencing the knowledge library or performing basic Google searches."

A Slice of Success

With the aid of Telerik RadControls for ASP.NET AJAX, the LabSlice team ultimately shaved ninety days off of its already aggressive development schedule, bringing its LabSlice.com application to market earlier than expected. Ellis credits the technology for simplifying and accelerating the development process, saving the team time, money, and effort.

"By using Telerik controls, our developers could concentrate on the key aspects of our product, and save their best efforts for the real technology challenges that are the focus of our startup," he says. "It ultimately allowed us to release LabSlice.com to market three months ahead of schedule, which generated tangible savings in costs."

LabSlice.com, which launched on September 1, 2010 has already begun attracting strong interest, and has been repeatedly lauded by users:

"Your service is brilliant in its simplicity and utility. I especially like how you've integrated AMIs with The Cloud Market – another outstanding tool for AWS Solution Providers."

"It's great to be able to deploy fully accessible product trials (web, database and console) to potential customers, simply with a couple of clicks in the LabSlice lab."

"Deploying a demo of a thick-client solution has never been easier!"

The application also garnered acclaim from AWS: LabSlice.com was named as the regional semi-finalist for Australia in the 2010 AWS Start-Up Challenge. The prestigious annual competition evaluates entrants from around the world on a variety of criteria, including originality and creativity, scalability, functionality, and implementation.

Ellis and the LabSlice team remain pleased by their continued success and look forward to future expansions of the application. They also are satisfied that their decision to go with Telerik RadControls for ASP.NET AJAX proved to be the right one.

Source: http://labslice.com/

7.4 Summary

- In this unit, you learned how to use one of ASP.NET's most practical features: validation.
- You saw how ASP.NET combines server-side and client-side validation to ensure bulletproof security without sacrificing the usability of your web pages.
- You also looked at the types of validation provided by the various validation controls, and even brushed up on the powerful pattern-matching syntax used for regular expressions.
- Finally, you considered how to customize and extend the validation process to handle a few different scenarios.

Notes

7.5 Keywords

Client/Server Architecture: An application architecture in which the server dispenses (or serves) information that is requested by one or more client applications.

Client-side: An operation or event that occurs on a client system. Examples include client-side scripting, client-side validation, and client-side events.

Custom Validator: Validation is performed by a user-defined function.

Validation Server Controls: A set of server controls, included with ASP.NET, that verify user input. The input is checked as it comes from HTML server controls and Web server controls (for example, a Web page form) against programmer-defined requirements. Validation controls perform input checking in server code. If the user is working with a browser that supports DHTML, the validation controls can also perform validation using client script.

7.6 Self Assessment

Fill in the blanks:

1. ASP.NET provides ..... validator controls.
2. ASP.NET automatically adds ..... code for client-side validation.
3. The validation controls are found in the System.Web.UI.WebControls namespace and inherit from the ..... class.
4. ..... is useful when you're placing several validators next to each other.
5. ASP.NET's most powerful validation controls is .....
6. All regular expressions consist of two kinds of characters: literals and .....

State whether the following statements are True or False:

7. Metacharacters provide the true secret to unlocking the full power of regular expressions.
8. Some logic is much more difficult to model in a regular expression.
9. Each validation control can be bound to a triple input control.
10. Every button has a CausesValidation property, which can be set to true or false.

7.7 Review Questions

1. What is the reason for validation?
2. What do you do if you want a button to post the page without checking validation?
3. What is the best type of validator to use for a radio button list?
4. What are the differences between the static and dynamic values of the display property?
5. Suppose the first item in your drop-down list is "Choose a payment method." How do you make sure users choose one?
6. What's the benefit of using the ValidationSummary control?
7. What control should you use to make sure the user can't order more of a single item than you actually have in stock?

Notes

8. Suppose you run a hotel that requires at least two guests stay in a double room, but no more than five guests. What control should you use on the "Number of guests" field?
9. How do you check that the user has entered a valid email address?
10. Suppose your theme park offers discounts to customers between the ages of 6 and 12, and also to customers over 65. What kind of control would you use to validate whether the customer is eligible for a discount, while still using a single age field?

Answers: Self Assessment

1.	five	2.	JavaScript
3.	BaseValidator	4.	Dynamic
5.	RegularExpressionValidator	6.	metacharacters
7.	True	8.	True
9.	False	10.	True

7.8 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Notes

Unit 8: Rich Controls

CONTENTS

Objectives

Introduction

8.1 The Calendar

8.1.1 Formatting the Calendar

8.1.2 Restricting Dates

8.2 The AdRotator

8.2.1 The Advertisement File

8.2.2 The AdRotator Class

8.3 Pages with Multiple Views

8.3.1 The MultiView Control

8.3.2 Creating Views

8.3.3 Showing a View

8.3.4 The Wizard Control

8.4 Summary

8.5 Keywords

8.6 Self Assessment

8.7 Review Questions

8.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Define calendar
- Describe AdRotator
- Explain pages with multiple view

Introduction

Among the many web controls available with ASP.NET 3.5 are a number of rich data controls. These controls are highly functional and have a wide selection of properties, events, and features that make them quite flexible and adaptable. Two such controls, the Calendar and AdRotator, feature both ease of use and flexibility. Each is briefly examined next for some typical uses.

In this unit, you'll take a look at several web controls that have no direct equivalent in the world of ordinary HTML. You'll start with the Calendar, which provides slick date-selection functionality. Next, you'll consider the AdRotator, which gives you an easy way to insert a randomly selected image into a web page. Finally, you'll learn how to create sophisticated pages with multiple

Notes

views using two advanced container controls: the MultiView and the Wizard. These controls allow you to pack a miniature application into a single page. Using them, you can handle a multistep task without redirecting the user from one page to another.

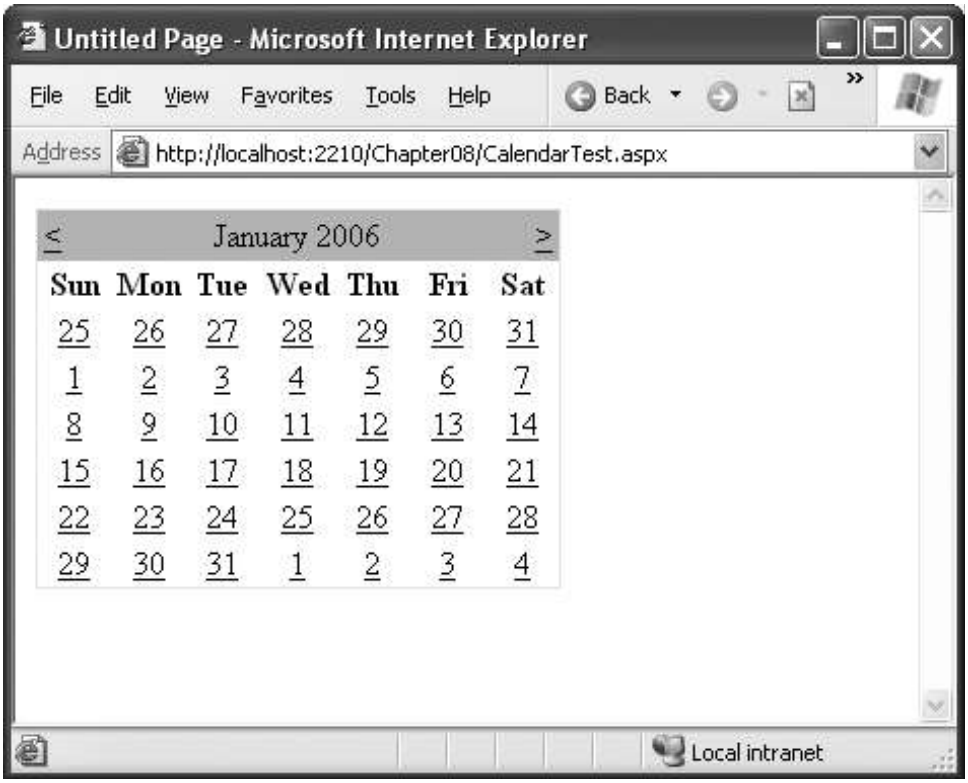
8.1 The Calendar

The Calendar control presents a miniature calendar that you can place in any web page. Like most rich controls, the Calendar can be programmed as a single object (and defined in a single simple tag), but it renders itself with dozens of lines of HTML output.

```
<asp:Calendar id="MyCalendar" runat="server" />
```

The Calendar control presents a single-month view, as shown in Figure 8.1. The user can navigate from month to month using the navigational arrows, at which point the page is posted back and ASP.NET automatically provides a new page with the correct month values.

Figure 8.1: The Default Calendar



You don't need to write any additional event-handling code to manage this process. When the user clicks a date, the date becomes highlighted in a gray box (by default). You can retrieve the selected day in your code as a DateTime object from the Calendar.SelectedDate property.

Notes ASP.NET includes numerous rich controls that are discussed elsewhere in this book, including rich data controls, security controls, and controls tailored for web portals. In this unit, you'll focus on a few useful web controls that don't fit neatly into any of these categories. All of these controls appear in the Standard tab of the Visual Studio Toolbox.

Notes

This basic set of features may provide everything you need in your application. Alternatively, you can configure different selection modes to allow users to select entire weeks or months or to render the control as a static calendar that doesn't allow selection. The only fact you must remember is that if you allow month selection, the user can also select a single week or a day. Similarly, if you allow week selection, the user can also select a single day.

You set the type of selection through the Calendar.SelectionMode property. You may also need to set the Calendar.FirstDayOfWeek property to configure how a week is selected.

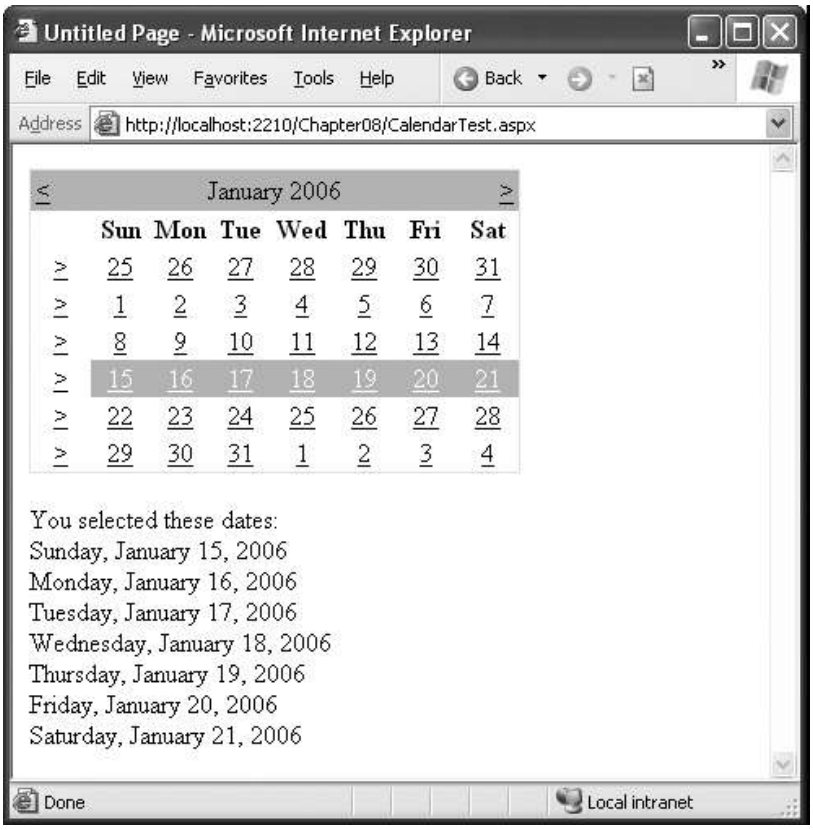
Task Set FirstDayOfWeek to the enumerated value Sunday, and weeks will be selected from Sunday to Saturday.

When you allow multiple date selection, you need to examine the SelectedDates property, which provides a collection of all the selected dates. You can loop through this collection using the foreach syntax. The following code demonstrates this technique:

```
lblDates.Text = "You selected these dates:<br />";
foreach (DateTime dt in MyCalendar.SelectedDates)
{
lblDates.Text += dt.ToLongDateString() + "<br />";
}
```

Figure 8.2 shows the resulting page after this code has been executed.

Figure 8.2: Selecting Multiple Dates



Notes

8.1.1 Formatting the Calendar

The Calendar control provides a whole host of formatting-related properties. You can set various parts of the calendar, like the header, selector, and various day types, by using one of the style properties (for example, WeekendDayStyle). Each of these style properties references a full-featured TableItemStyle object that provides properties for coloring, border style, font, and alignment. Taken together, they allow you to modify almost any part of the calendar's appearance.

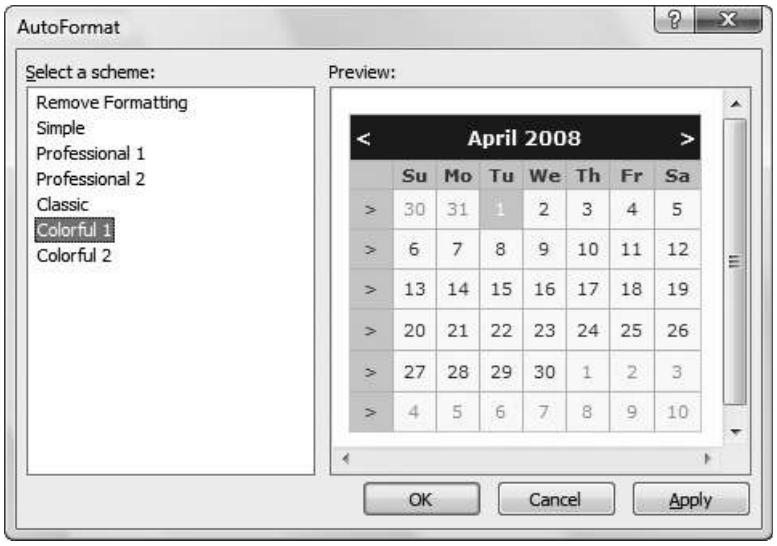
Table 8.1 lists the style properties that the Calendar control provides.

Table 8.1: Properties for Calendar Styles

Member	Description
DayHeaderStyle	The style for the section of the Calendar that displays the days of the week (as column headers).
DayStyle	The default style for the dates in the current month.
NextPrevStyle	The style for the navigation controls in the title section that move from month to month.
OtherMonthDayStyle	The style for the dates that aren't in the currently displayed month. These dates are used to "fill in" the calendar grid. For example, the first few cells in the topmost row may display the last few days from the previous month.
SelectedDayStyle	The style for the selected dates on the calendar.
SelectorStyle	The style for the week and month date-selection controls.
TitleStyle	The style for the title section.
TodayDayStyle	The style for the date designated as today (represented by the TodaysDate property of the Calendar control).
WeekendDayStyle	The style for dates that fall on the weekend.

You can adjust each style using the Properties window. For a quick shortcut, you can set an entire related color scheme using the Calendar's Auto Format feature. To do so, start by selecting the Calendar on the design surface of a web form. Then, click the arrow icon that appears next to its top-right corner to show the Calendar's smart tag, and click the Auto Format link. You'll be presented with a list of predefined formats that set the style properties, as shown in Figure 8.3.

Figure 8.3: Calendar Styles



Notes

You can also use additional properties to hide some elements or configure the text they display.

Example: Properties that start with "Show" (such as ShowDayHeader, ShowTitle, and ShowGridLines) can be used to hide or show a specific visual element. Properties that end in "Text" (such as PrevMonthText, NextMonthText, and SelectWeekText) allow you to set the text that's shown in part of the calendar.

8.1.2 Restricting Dates

In most situations where you need to use a calendar for selection, you don't want to allow the user to select any date in the calendar. For example, the user might be booking an appointment or choosing a delivery date two services that are generally provided only on set days. The Calendar control makes it surprisingly easy to implement this logic. In fact, if you've worked with the date and time controls on the Windows platform, you'll quickly recognize that the ASP.NET versions are far superior.

The basic approach to restricting dates is to write an event handler for the Calendar.DayRender event. This event occurs when the Calendar control is about to create a month to display to the user. This event gives you the chance to examine the date that is being added to the current month (through the e.Day property) and decide whether it should be selectable or restricted.

The following code makes it impossible to select any weekend days or days in years greater than 2010:

```
protected void MyCalendar_DayRender(Object source, DayRenderEventArgs e)
{
// Restrict dates after the year 2010 and those on the weekend.
if (e.Day.IsWeekend || e.Day.Date.Year > 2010)
{
e.Day.IsSelectable = false;
}
}
```

The e.Day object is an instance of the CalendarDay class, which provides various properties. Table 8.2 describes some of the most useful.

Table 8.2: CalendarDay Properties

Property	Description
Date	The DateTime object that represents this date.
IsWeekend	True if this date falls on a Saturday or Sunday.
IsToday	True if this value matches the Calendar.TodaysDate property, which is set to the current day by default.
IsOtherMonth	True if this date doesn't belong to the current month but is displayed to fill in the first or last row. For example, this might be the last day of the previous month or the next day of the following month.
IsSelectable	Allows you to configure whether the user can select this day.

The DayRender event is extremely powerful. Besides allowing you to tailor what dates are selectable, it also allows you to configure the cell where the date is located through the e.Cell property. (The calendar is displayed using an HTML table.)

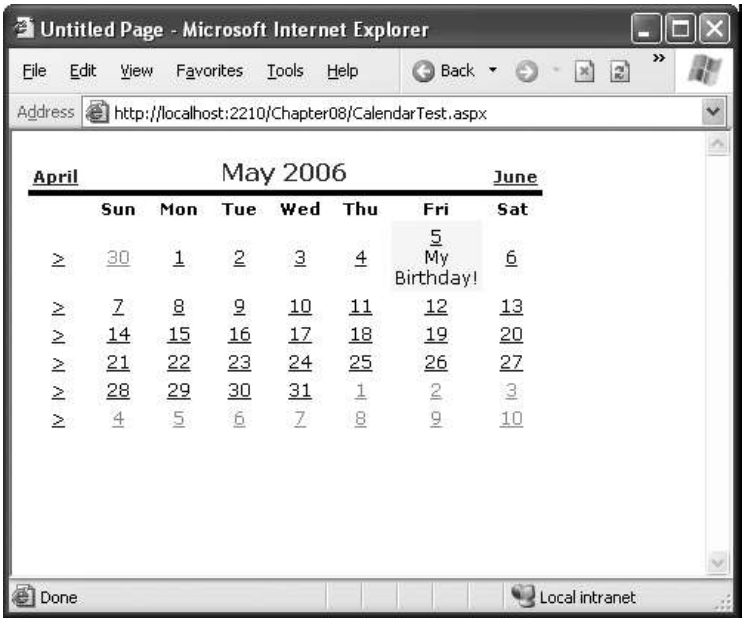
Notes

Example: You could highlight an important date or even add information. Here's an example that highlights a single day the fifth of May by adding a new Label control in the table cell for that day:

```
protected void MyCalendar_DayRender(Object source, DayRenderEventArgs e)
{
// Check for May 5 in any year, and format it.
if (e.Day.Date.Day == 5 && e.Day.Date.Month == 5)
{
e.Cell.BackColor = System.Drawing.Color.Yellow;
// Add some static text to the cell.
Label lbl = new Label();
lbl.Text = "<br />My Birthday!";
e.Cell.Controls.Add(lbl);
}
}
```

Figure 8.4 shows the resulting calendar display.

Figure 8.4: Highlighting a Day



The Calendar control provides two other useful events: SelectionChanged and Visible Month Changed. These occur immediately after the user selects a new day or browses to a new month (using the next month and previous month links). You can react to these events and update other portions of the web page to correspond to the current calendar month. For example, you could design a page that lets you schedule a meeting in two steps. First, you choose the appropriate day. Then, you choose one of the available times on that day.

The following code demonstrates this approach, using a different set of time values if a Monday is selected in the calendar than it does for other days:

```
protected void MyCalendar_SelectionChanged(Object source, EventArgs e)
{
```

Notes

```
lstTimes.Items.Clear();
switch (MyCalendar.SelectedDate.DayOfWeek)
{
case DayOfWeek.Monday:
// Apply special Monday schedule.
lstTimes.Items.Add("10:00");
lstTimes.Items.Add("10:30");
lstTimes.Items.Add("11:00");
break;
default:
lstTimes.Items.Add("10:00");
lstTimes.Items.Add("10:30");
lstTimes.Items.Add("11:00");
lstTimes.Items.Add("11:30");
lstTimes.Items.Add("12:00");
lstTimes.Items.Add("12:30");
break;
}
}
```

To try these features of the Calendar control, run the Appointment.aspx page from the online samples. This page provides a formatted Calendar control that restricts some dates, formats others specially, and updates a corresponding list control when the selection changes.

Table 8.3 gives you an at-a-glance look at almost all the members of the Calendar control class.

Table 8.3: CalendarMembers

Member	Description
Caption and CaptionAlign	Gives you an easy way to add a title to the calendar. By default, the caption appears at the top of the title area, just above the month heading. However, you can control this to some extent with the CaptionAlign property. Use Left or Right to keep the caption at the top but move it to one side or the other, and use Bottom to place the caption under the calendar.
CellPadding	ASP.NET creates a date in a separate cell of an invisible table. CellPadding is the space, in pixels, between the border of each cell and its contents.
CellSpacing	The space, in pixels, between cells in the same table.
DayNameFormat	Determines how days are displayed in the calendar header. Valid values are Full (as in Sunday), FirstLetter (S), FirstTwoLetters (Su), and Short (Sun), which is the default.
FirstDayOfWeek	Determines which day is displayed in the first column of the calendar. The values are any day name from the FirstDayOfWeek enumeration (such as Sunday).
NextMonthText and PrevMonthText	Sets the text that the user clicks to move to the next or previous month. These navigation links appear at the top of the calendar and are the greater-than (>) and less-than (<) signs by default. This setting is applied only if NextPrevFormat is set to CustomText.
NextPrevFormat	Sets the text that the user clicks to move to the next or previous month. This can be FullMonth (for example, December), ShortMonth (Dec), or CustomText, in which case the NextMonthText and PrevMonthText properties are used. CustomText is the default.

Contd...

Notes

SelectedDate and SelectedDates	Sets or gets the currently selected date as a DateTime object. You can specify this in the control tag in a format like this: "12:00:00 AM, 12/31/2010" (depending on your computer's regional settings). If you allow multiple date selection, the SelectedDates property will return a collection of DateTime objects, one for each selected date. You can use collection methods such as Add, Remove, and Clear to change the selection.
SelectionMode	Determines how many dates can be selected at once. The default is Day, which allows one date to be selected. Other options include DayWeek (a single date or an entire week) or DayWeekMonth (a single date, an entire week, or an entire month). You have no way to allow the user to select multiple noncontiguous dates. You also have no way to allow larger selections without also including smaller selections. (For example, if you allow full months to be selected, you must also allow week selection and individual day selection.)
SelectMonthText and SelectWeekText	The text shown for the link that allows the user to select an entire month or week. These properties don't apply if the SelectionMode is Day.
ShowDayHeader, ShowGridLines, ShowNextPrevMonth, and ShowTitle	These Boolean properties allow you to configure whether various parts of the calendar are shown, including the day titles, gridlines between every day, the previous/next month navigation links, and the title section. Note that hiding the title section also hides the next and previous month navigation controls.
TitleFormat	Configures how the month is displayed in the title area. Valid values include Month and MonthYear (the default).
TodaysDate	Sets which day should be recognized as the current date and formatted with the TodayDayStyle. This defaults to the current day on the web server.
VisibleDate	Gets or sets the date that specifies what month will be displayed in the calendar. This allows you to change the calendar display without modifying the current date selection.
DayRender event	Occurs once for each day that is created and added to the currently visible month before the page is rendered. This event gives you the opportunity to apply special formatting, add content, or restrict selection for an individual date cell. Keep in mind that days can appear in the calendar even when they don't fall in the current month, provided they fall close to the end of the previous month or close to the start of the following month.
SelectionChanged event	Occurs when the user selects a day, a week, or an entire month by clicking the date selector controls.
VisibleMonthChanged event	Occurs when the user clicks the next or previous month navigation controls to move to another month.

8.2 The AdRotator

The basic purpose of the AdRotator is to provide a graphic on a page that is chosen randomly from a group of possible images. In other words, every time the page is requested, an image is selected at random and displayed, which is the "rotation" indicated by the name AdRotator. One use of the AdRotator is to show banner-style advertisements on a page, but you can use it any time you want to vary an image randomly.

Using ASP.NET, it wouldn't be too difficult to implement an AdRotator type of design on your own. You could react to the Page.Load event, generate a random number, and then use that number to choose from a list of predetermined image files. You could even store the list in the web.config file so that it can be easily modified separately as part of the application's configuration. Of course, if you wanted to enable several pages with a random image, you would either have to repeat the code or create your own custom control. The AdRotator provides these features for free.

Notes

8.2.1 The Advertisement File

The AdRotator stores its list of image files in an XML file. This file uses the format shown here:

```
<Advertisements>
<Ad>
<ImageUrl>prosetech.jpg</ImageUrl>
<NavigateUrl>http://www.prosetech.com</NavigateUrl>
<AlternateText>ProseTech Site</AlternateText>
<Impressions>1</Impressions>
<Keyword>Computer</Keyword>
</Ad>
</Advertisements>
```

This example shows a single possible advertisement. To add more advertisements, you would create multiple <Ad> elements and place them all inside the root <Advertisements> element:

```
<Advertisements>
<Ad>
<!-- First ad here. -->
</Ad>

<Ad>
<!-- Second ad here. -->
</Ad>
</Advertisements>
```

Each <Ad> element has a number of other important properties that configure the link, the image, and the frequency, as described in Table 8.4.

Table 8.4: Advertisement File Elements

Elements	Description
ImageUrl	The image that will be displayed. This can be a relative link (a file in the current directory) or a fully qualified Internet URL.
NavigateUrl	The link that will be followed if the user clicks the banner.
AlternateText	The text that will be displayed instead of the picture if it cannot be displayed. This text will also be used as a tooltip in some newer browsers.
Impressions	A number that sets how often an advertisement will appear. This number is relative to the numbers specified for other ads. For example, a banner with the value 10 will be shown twice as often (on average) as the banner with the value 5.
Keyword	A keyword that identifies a group of advertisements. You can use this for filtering. For example, you could create ten advertisements and give half of them the keyword Retail and the other half the keyword Computer. The web page can then choose to filter the possible advertisements to include only one of these groups.

8.2.2 The AdRotator Class

The actual AdRotator class provides a limited set of properties. You specify both the appropriate advertisement file in the AdvertisementFile property and the type of window that the link should follow (the Target window). The target can name a specific frame, or it can use one of the values defined in Table 8.5.

Notes

Table 8.5: Special Frame Targets

Target	Description
_blank	The link opens a new unframed window.
_parent	The link opens in the parent of the current frame.
_self	The link opens in the current frame.
_top	The link opens in the topmost frame of the current window (so the link appears in the full window).

Optionally, you can set the KeywordFilter property so that the banner will be chosen from a specific keyword group. This is a fully configured AdRotator tag:

```
<asp:AdRotator id="Ads" runat="server" AdvertisementFile="MainAds.xml"
Target="_blank" KeywordFilter="Computer" />
```

Additionally, you can react to the AdRotator.AdCreated event. This occurs when the page is being created and an image is randomly chosen from the advertisements file. This event provides you with information about the image that you can use to customize the rest of your page. For example, you might display some related content or a link, as shown in Figure 8.5.

Figure 8.5: An AdRotator with Synchronized Content

The event-handling code for this example simply configures a HyperLink control named lnkBanner based on the randomly selected advertisement:

```
protected void Ads_AdCreated(Object sender, AdCreatedEventArgs e)
{
// Synchronize the Hyperlink control.
lnkBanner.NavigateUrl = e.NavigateUrl;
// Syncrhonize the text of the link.
lnkBanner.Text = "Click here for information about our sponsor: ";
lnkBanner.Text += e.AlternateText;
}
```

As you can see, rich controls such as the Calendar and AdRotator don't just add a sophisticated HTML output, they also include an event framework that allows you to take charge of the control's behavior and integrate it into your application.

Notes

8.3 Pages with Multiple Views

In a typical website, you'll surf through many separate pages. For example, if you want to add an item to your shopping cart and take it to the checkout in an e-commerce site, you'll need to jump from one page to another. This design has its advantages—namely, it lets you carefully separate different tasks into different code files. It also presents some challenges; for example, you need to come up with a way to transfer information from one page to another.

However, in some cases it makes more sense to create a single page that can handle several different tasks. For example, you might want to provide several views of the same data (such as a grid-based view and a chart-based view) and allow the user to switch from one view to the other without leaving the page. Or, you might want to handle a small multistep task in one place (such as supplying user information for an account sign-up process). In these examples, you need a way to create dynamic pages that provide more than one possible view. Essentially, the page hides and shows different controls depending on which view you want to present.

The simplest way to understand this technique is to create a page with several Panel controls. Each panel can hold a group of ASP.NET controls. For example, imagine you're creating a simple three-step wizard. You'll start by adding three panels to your page, one for each step say, panelStep1, panelStep2, and panelStep3. You can place the panels one after the other, because you'll show only one at a time. Once you've added the panels, you can place the appropriate controls inside each panel. To start, the Visible property of each panel should be false, except for panelStep1, which appears the first time the user requests the page.

Here's an example that shows the way you can arrange your panels:

```
<asp:Panel ID="panelStep1" runat="server">...</asp:Panel>
<asp:Panel ID="panelStep2" Visible="False" runat="server">...</asp:Panel>
<asp:Panel ID="panelStep3" Visible="False" runat="server">...</asp:Panel>
```

Notes When you set the Visible property of a control to false, the control won't appear in the page at runtime. Any controls inside an invisible panel are also hidden from sight, and they won't be present in the rendered HTML for the page. However, these controls will still appear in the Visual Studio design surface so that you can still select them and configure them.

Finally, you'll add one or more navigation buttons outside the panels. For example, the following code handles the click of a Next button, which is placed just after panelStep3 (so it always appears at the bottom of the page). The code checks which step the user is currently on, hides the current panel, and shows the following panel. This way the user is moved to the next step.

```
protected void cmdNext_Click(object sender, EventArgs e)
{
if (panelStep1.Visible)
{
// Move to step 2.
panelStep1.Visible = false;
panelStep2.Visible = true;
}
else if (panelStep2.Visible)
{
```

Notes

```
// Move to step 3.
panelStep2.Visible = false;
panelStep3.Visible = true;
// Change text of button from Next to Finish.
cmdNext.Text = "Finish";
}
else if (panelStep3.Visible)
{
// The wizard is finished.
panelStep3.Visible = false;
// Add code here to perform the appropriate task
// with the information you've collected.
}
}
```

This approach works relatively well. Even when the panels are hidden, you can still interact with all the controls on each panel and retrieve the information they contain. The problem is that you need to write all the code for controlling which panel is visible. If you make your wizard much more complex for example, you want to add a button for returning to a previous step it becomes more difficult to keep track of what's happening. At best, this approach clutters your page with the code for managing the panels. At worst, you'll make a minor mistake and end up with two panels showing at the same time.

Fortunately, ASP.NET gives you a more robust option. You can use two controls that are designed for the job the MultiView and the Wizard.

8.3.1 The MultiView Control

TheMultiView is the simpler of the two multiple-view controls. Essentially, the MultiView gives you a way to declare multiple views and show only one at a time. It has no default user interface you get only whatever HTML and controls you add. The MultiView is equivalent to the custom panel approach explained earlier.

Creating a MultiView is suitably straightforward. You add the <asp:MultiView> tag to your .aspx page file and then add one <asp:View> tag inside it for each separate view:

```
<asp:MultiView ID="MultiView1" runat="server">
<asp:View ID="View1" runat="server">...</asp:View>
<asp:View ID="View2" runat="server">...</asp:View>
<asp:View ID="View3" runat="server">...</asp:View>
</asp:MultiView>
```

In Visual Studio, you create these tags by first dropping a MultiView control onto your form and then using the Toolbox to add as many View controls inside it as you want. This drag-and-drop process can be a bit tricky. When you add the first View control, you must make sure to drop it in the blank area inside the MultiView (not next to the MultiView, or on the MultiView's title bar). When you add more View controls, you must drop each one on one of the gray header bars of one of the existing views. The gray header has the View title (such as "View1" or "View2").

The View control plays the same role as the Panel control in the previous example, and the MultiView takes care of coordinating all the views so that only one is visible at a time.

Notes

8.3.2 Creating Views

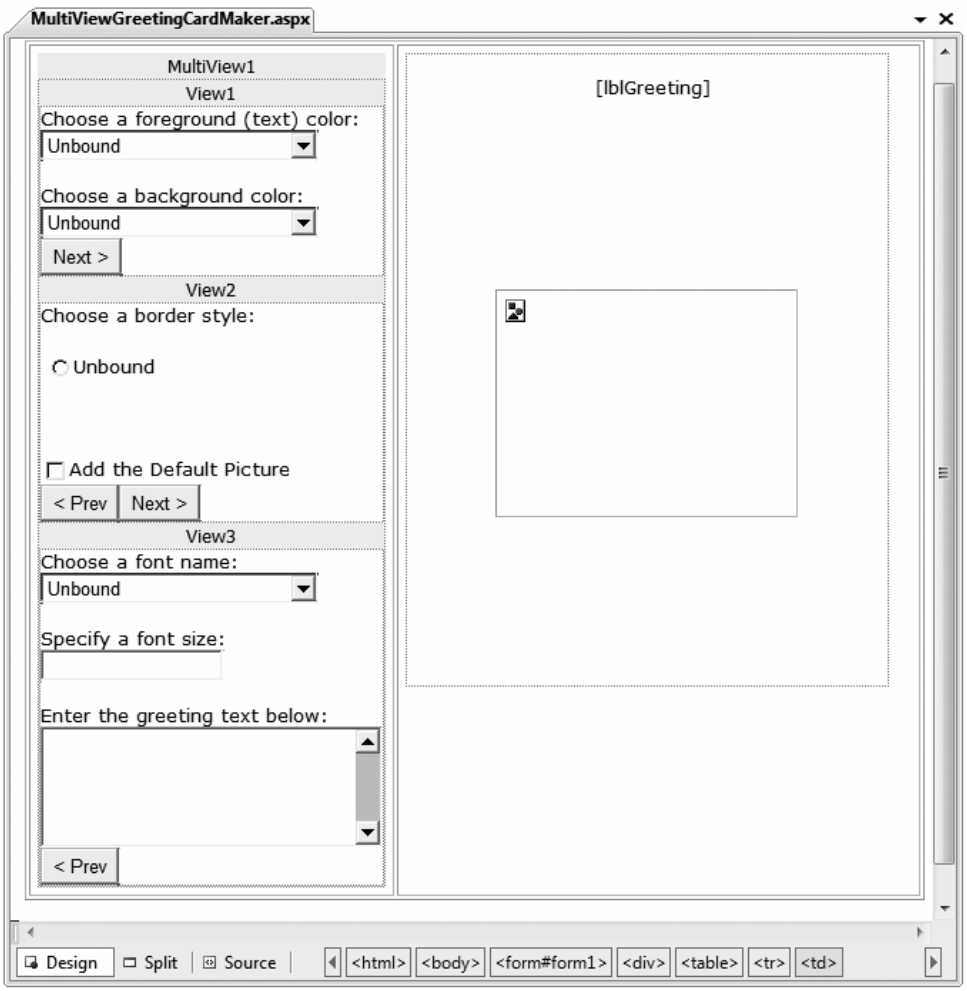
Here's the full markup for a MultiView that splits the greeting card controls into three views named View1, View2, and View3:

```
<asp:MultiView id="MultiView1" runat="server" >

<asp:View ID="View1" runat="server">
Choose a foreground (text) color:<br />
<asp:DropDownList ID="lstForeColor" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="ControlChanged" />
<br /><br />
Choose a background color:<br />
<asp:DropDownList ID="lstBackColor" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="ControlChanged" />
</asp:View>

<asp:View ID="View2" runat="server">
Choose a border style:<br />
<asp:RadioButtonList ID="lstBorder" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="ControlChanged" RepeatColumns="2" />
<br />
<asp:CheckBox ID="chkPicture" runat="server" AutoPostBack="True"
OnCheckedChanged="ControlChanged" Text="Add the Default Picture" />
</asp:View>

<asp:View ID="View3" runat="server">
Choose a font name:<br />
<asp:DropDownList ID="lstFontName" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="ControlChanged" />

<br /><br />
Specify a font size:<br />
<asp:TextBox ID="txtFontSize" runat="server" AutoPostBack="True"
OnTextChanged="ControlChanged" />
<br /><br />
Enter the greeting text below:<br />
<asp:TextBox ID="txtGreeting" runat="server" AutoPostBack="True"
OnTextChanged="ControlChanged" TextMode="MultiLine" />
</asp:View>

</asp:MultiView>
```

Visual Studio shows all your views at design time, one after the other (Figure 8.6). You can edit these regions in the same way you design any other part of the page.

Notes

Figure 8.6: Designing Multiple Views



8.3.3 Showing a View

If you run this example, you won't see what you expect. The MultiView will appear empty on the page, and all the controls in all your views will be hidden.

The reason this happens is because the MultiView.ActiveViewIndex property is, by default, set to –1. The ActiveViewIndex property determines which view will be shown. If you set the ActiveViewIndex to 0, however, you'll see the first view. Similarly, you can set it to 1 to show the second view, and so on. You can set this property using the Properties window or using code:

```
// Show the first view.
MultiView1.ActiveViewIndex = 0;
```

This example shows the first view (View1) and hides whatever view is currently being displayed, if any.

You can also use the SetActiveView() method, which accepts any one of the view objects you've created. This may result in more readable code (if you've chosen descriptive IDs for your view controls), and it ensures that any errors are caught earlier (at compile time instead of runtime).

```
MultiView1.SetActiveView(View1);
```

Notes

This gives you enough functionality that you can create previous and next navigation buttons. However, it's still up to you to write the code that checks which view is visible and changes the view. This code is a little simpler, because you don't need to worry about hiding views any longer, but it's still less than ideal.

Fortunately, the MultiView includes some built-in smarts that can save you a lot of trouble. Here's how it works: the MultiView recognizes button controls with specific command names. (Technically, a button control is any control that implements the IButtonControl interface, including the Button, ImageButton, and LinkButton.) If you add a button control to the view that uses one of these recognized command names, the button gets some automatic functionality. Using this technique, you can create navigation buttons without writing any code.

Table 8.6 lists all the recognized command names. Each command name also has a corresponding static field in the MultiView class, so you can easily get the right command name if you choose to set it programmatically.

Table 8.6: Recognized Command Names for the MultiView

Command Name	MultiView Field	Description
PrevView	PreviousViewCommandName	Moves to the previous view.
NextView	NextViewCommandName	Moves to the next view.
SwitchViewByID	SwitchViewByIDCommandName	Moves to the view with a specific ID (string name). The ID is taken from the CommandArgument property of the button control.
SwitchViewByIndex	SwitchViewByIndexCommandName	Moves to the view with a specific numeric index. The index is taken from the CommandArgument property of the button control.

To try this, add this button to the first view:

```
<asp:Button ID="Button1" runat="server" CommandArgument="View2"
CommandName="SwitchViewByID" Text="Go to View2" />
```

When clicked, this button sets the MultiView to show the view specified by the CommandArgument (View2).

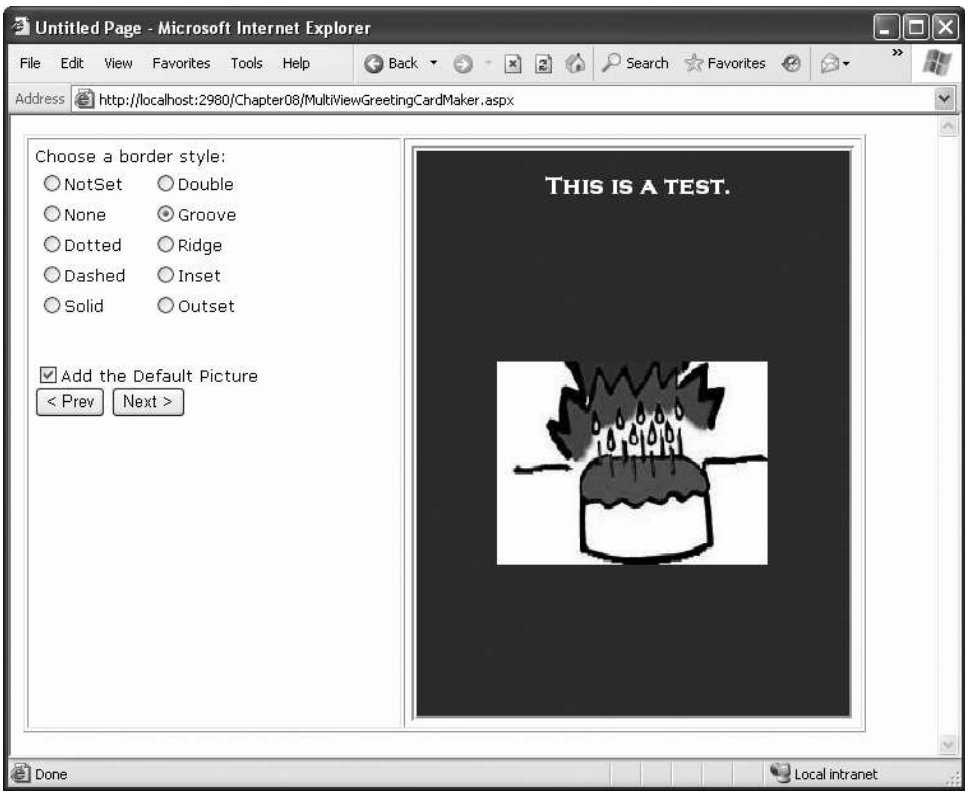
Rather than create buttons that take the user to a specific view, you might want a button that moves forward or backward one view. To do this, you use the PrevView and NextView command names. Here's an example that defines previous and next buttons in the second View:

```
<asp:Button ID="Button1" runat="server" Text="< Prev" CommandName="PrevView"
/>
<asp:Button ID="Button2" runat="server" Text="Next >" CommandName="NextView"
/>
```

Once you add these buttons to your view, you can move from view to view easily. Figure 8.7 shows the previous example with the second view currently visible.

Notes

Figure 8.7: Moving from one view to another



Task Write a code to highlight an important date or even add information.

8.3.4 The Wizard Control

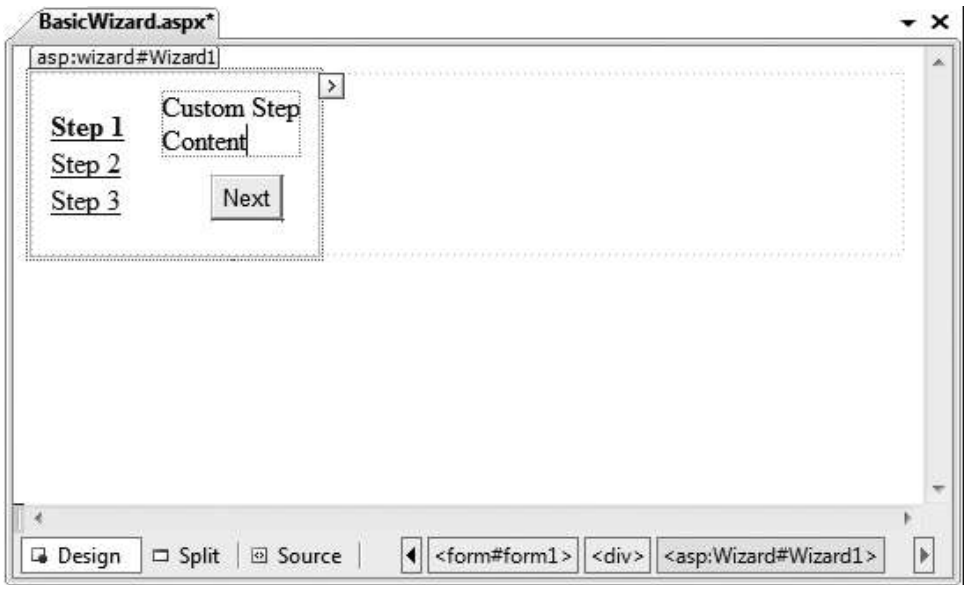
The Wizard control is a more glamorous version of the MultiView control. It also supports showing one of several views at a time, but it includes a fair bit of built-in yet customizable behavior, including navigation buttons, a sidebar with step links, styles, and templates.

Usually, wizards represent a single task, and the user moves linearly through them, moving from the current step to the one immediately following it (or the one immediately preceding it in the case of a correction). The ASP.NET Wizard control also supports nonlinear navigation, which means it allows you to decide to ignore a step based on the information the user supplies.

By default, the Wizard control supplies navigation buttons and a sidebar with links for each step on the left. You can hide the sidebar by setting the Wizard.DisplaySideBar property to false. Usually, you'll take this step if you want to enforce strict step-by-step navigation and prevent the user from jumping out of sequence. You supply the content for each step using any HTML or ASP.NET controls. Figure 8.8 shows the region where you can add content to an out-of-the-boxWizard instance.

Notes

Figure 8.8: The Region for Step Content



Wizard Steps

To create a wizard in ASP.NET, you simply define the steps and their content using <asp:WizardStep> tags. Here's the basic structure you'll use:

```
<asp:Wizard ID="Wizard1" runat="server" ... >
<WizardSteps>

<asp:WizardStep runat="server" Title="Step 1 ">
...
</asp:WizardStep>

<asp:WizardStep runat="server" Title="Step 1 ">
...
</asp:WizardStep>

...
<WizardSteps>
</asp:Wizard>
```

You can add as many WizardStep controls inside the Wizard as you want. Conceptually, the WizardStep plays the same role as the View in a MultiView (or the basic Panel in the first example that you considered). You place the content for each step inside the WizardStep control.

Before you start adding the content to your wizard, it's worth reviewing Table 8.7, which shows a few basic pieces of information that you can define for each step.

Notes

Table 8.7: WizardStep Properties

Property	Description
Title	The descriptive name of the step. This name is used for the text of the links in the sidebar.
StepType	The type of step, as a value from the WizardStepType enumeration. This value determines the type of navigation buttons that will be shown for this step. Choices include Start (shows a Next button), Step (shows Next and Previous buttons), Finish (shows Finish and Previous buttons), Complete (shows no buttons and hides the sidebar, if it's enabled), and Auto (the step type is inferred from the position in the collection). The default is Auto, which means the first step is Start, the last step is Finish, and all other steps are Step.
AllowReturn	Indicates whether the user can return to this step. If false, once the user has passed this step, the user will not be able to return. The sidebar link for this step will have no effect, and the Previous button of the following step will either skip this step or be hidden completely (depending on the AllowReturn value of the preceding steps).

To see how this works, consider a wizard that again uses the GreetingCardMaker example. It guides the user through four steps. The first three steps allow the user to configure the greeting card, and the final step shows the generated card. The entire process is shown in Figure 8.9.

```
<asp:Wizard ID="Wizard1" runat="server" ActiveStepIndex="0"
BackColor="LemonChiffon" BorderStyle="Groove" BorderWidth="2px"
CellPadding="10">

<WizardSteps>
<asp:WizardStep runat="server" Title="Step 1 - Colors">
Choose a foreground (text) color:<br />
<asp:DropDownList ID="lstForeColor" runat="server" />
<br />
Choose a background color:<br />
<asp:DropDownList ID="lstBackColor" runat="server" />
</asp:WizardStep>

<asp:WizardStep runat="server" Title="Step 2 - Background">
Choose a border style:<br />
<asp:RadioButtonList ID="lstBorder" runat="server" RepeatColumns="2" />
<br /><br />
<asp:CheckBox ID="chkPicture" runat="server"
Text="Add the Default Picture" />
</asp:WizardStep>

<asp:WizardStep runat="server" Title="Step 3 - Text">
Choose a font name:<br />
<asp:DropDownList ID="lstFontName" runat="server" />
<br /><br />
Specify a font size:<br />
<asp:TextBox ID="txtFontSize" runat="server" />
<br /><br />
```

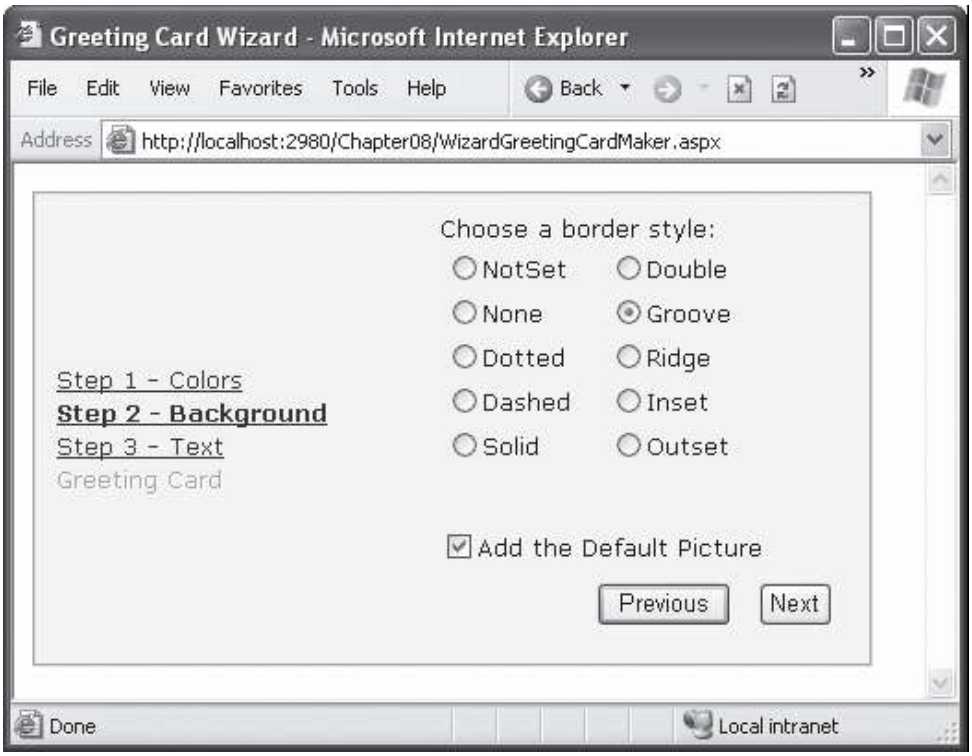
Notes

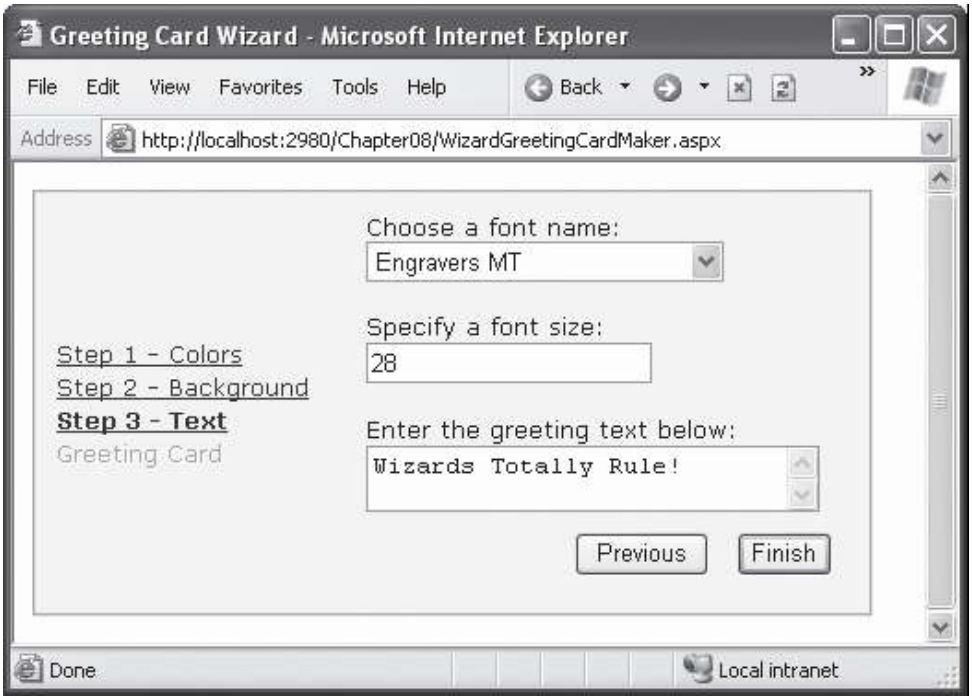
```
Enter the greeting text below:<br />
<asp:TextBox ID="txtGreeting" runat="server"
TextMode="MultiLine" />
</asp:WizardStep>

<asp:WizardStep runat="server" StepType="Complete" Title="Greeting Card">
<asp:Panel ID="pnlCard" runat="server" HorizontalAlign="Center">
<br />
<asp:Label ID="lblGreeting" runat="server" />
<asp:Image ID="imgDefault" runat="server" Visible="False" />
</asp:Panel>
</asp:WizardStep>
</WizardSteps>
</asp:Wizard>
```

Figure 8.9: A Wizard with four Steps

Contd...

Notes


Greeting Card Wizard - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address http://localhost:2980/Chapter08/WizardGreetingCardMaker.aspx
Choose a border style:
NotSet
Double
None
Groove
Dotted
Ridge
Dashed
Inset
Solid
Outset
Step 1 - Colors
Step 2 - Background
Step 3 - Text
Greeting Card
Add the Default Picture
Previous
Next
Done
Local intranet


Greeting Card Wizard - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address http://localhost:2980/Chapter08/WizardGreetingCardMaker.aspx
Choose a font name:
Engravers MT
Specify a font size:
28
Step 1 - Colors
Step 2 - Background
Step 3 - Text
Greeting Card
Enter the greeting text below:
Wizards Totally Rule!
Previous
Finish
Done
Local intranet

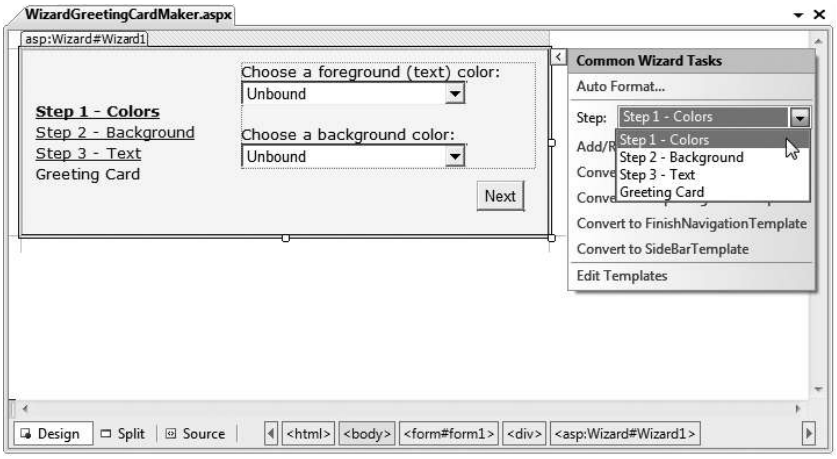
Contd...

Notes

If you look carefully, you'll find a few differences from the original page and the MultiView-based example. First, the controls aren't set to automatically post back. That's because the greeting card isn't rendered until the final step, at the conclusion of the wizard. Another change is that no navigation buttons exist. That's because the wizard adds these details automatically based on the step type. For example, you'll get a Next button for the first two steps, a Previous button for steps 2 and 3, and a Finish button for step 3. The final step, which shows the complete card, doesn't provide any navigation links because the StepType is set to Complete.

Unlike the MultiView control, you can see only one step at a time in Visual Studio. To choose which step you're currently designing, select it from the smart tag, as shown in Figure 8.10. But be warned every time you do, Visual Studio changes the Wizard. ActiveStepIndex property to the step you choose. Make sure you set this back to 0 before you run your application so it starts at the first step.

Figure 8.10: Designing a Step



Notes

Notes Remember, when you add controls to separate steps on a wizard, the controls are all instantiated and persisted in view state, regardless of which step is currently shown. If you need to slim down a complex wizard, you'll need to split it into separate pages, use the Server.Transfer() method to move from one page to the next, and tolerate a less elegant programming model.

Case Study

Mercury Payment Systems' Virtual Terminal

Merchant demand for safe, "anytime, anywhere" payment processing is on the rise. Mercury Payment Systems®, one of the fastest-growing payment processing companies in North America wanted to offer small to medium-sized businesses that don't yet need a POS, ECR, or terminal the ability to process secure transactions from any location with an internet connection.

"We wanted a reliable solution for merchants not quite ready for a full point-of-sale (POS) system, yet who needed an efficient and cost-effective means for processing card-based and mobile transactions," says Emily Jimmerson, Software Engineer, Mercury Payment Systems. "We take card holder data security very seriously, so our application would have to be fully compliant with current Payment Card Industry (PCI) standards, as well."

The company called upon its internal software development team to take Mercury VirtualTerminal from concept to reality. Developers were asked to create a flexible, easy-to-use internet-based card payment processing software application, which it planned to offer free of charge to customers. Mercury VirtualTerminal would have to support mobile transactions made via smartphones, be compatible with multiple browser technologies, and maintain PCI compliance. And, it would have to be completed within a timeframe of six months.

With rigorous project requirements and a demanding schedule, the team needed a way – an ace in the hole – to accelerate their development process.

Upping the Ante

Jimmerson and other Mercury developers began searching for tools and technologies that would help them expedite the creation and delivery of Mercury VirtualTerminal. After carefully weighing their options and with the clock ticking, the team decided to take a gamble on Telerik's RadControls for ASP.NET AJAX.

Once immersed in development, the decision to use RadControls began immediately paying off. RadWindow, RadFormDecorator, RadTabStrip, and RadToolTipManager proved to be indispensible in surmounting a variety of obstacles.

"One of our challenges was to display a printable, nicely formatted receipt showing transaction information. We experimented with opening a new tab or browser window, but that wasn't very user friendly," says Jimmerson. "With RadWindow, we were able to display the data in a clean container that could be easily minimized or dragged around the screen."

Contd...

Notes

RadFormDecorator and RadTabStrip played key roles as well, saving the team from spending hours designing custom CSS and skins. Developers initially relied on customized JQuery to achieve the application's tabbed layout, which was the most ideal way of displaying critical information. Although this approach was functionally successful, the results didn't have a professional look-and-feel. Upon migrating to RadTabStrip and leveraging customizable skins from RadFormDecorator, the team swiftly attained the crisp, consistent appearance that the Mercury VirtualTerminal needed.

"Customizing a web application with CSS and skins can be tricky; it's time-consuming to get everything just right," notes Jimmerson. "Telerik's controls looked great out-of-the-box and included pre-made skins that were easily applied project-wide."

Based on the nature of the market and customers it serves, Mercury VirtualTerminal contains numerous input fields. While some of the fields are self-explanatory, many required detailed tooltips explaining the type or format of data that could be entered. Rather than accepting bland, default tooltips, developers used RadToolTipManager to produce nicely styled tooltips that extend and match the application's polished look-and-feel.

Jimmerson's team also made good use of Telerik's world-class support system and extensive resource library.

"One major benefit came from using the Telerik support tickets," she says. "We chose the 24-hour response time option, and the help we received came with excellent code examples, links to external resources, and fully functional example projects. Our development time was reduced considerably as a result of the high quality help we received when troubleshooting issues."

Mercury Payment Systems Hits the Jackpot

Just as it was originally envisioned, Mercury VirtualTerminal is a flexible, intuitive internet-based, PCI-compliant card and mobile payment processing solution. Free and easy to use, it enables merchants to use any internet-enabled computer to login and process transactions using either a magnetic strip reader or by keying card numbers in manually.

After completing the application well under its six-month deadline, the team decided to let the chips fall where they may, unleashing Mercury VirtualTerminal early upon a small beta user group. The response took the company by surprise.

"In only three months, we far surpassed our goals for transaction processing volume," says Jimmerson.

With strongly positive reactions from beta users, Mercury VirtualTerminal was rolled out to the company's entire customer base a full two weeks ahead of schedule. Jimmerson and the developers credit their decision to try Telerik RadControls with the project's success. Recognizing that it would have taken significantly longer to develop working solutions that Telerik makes immediately available out of the box, Mercury Payment Systems has since gone on to deploy Telerik's controls in multiple other projects.

"We chose to go with Telerik because it could significantly speed up development, allowing us to meet our tight deadlines," said Jimmerson. "RadControls saved us time because we didn't need to create custom controls from scratch. It truly helped us meet our goal of producing a high-quality, functional application that remains user-friendly and has a professional layout."

Source: Mercury Payment Systems

Notes

8.4 Summary

- This unit showed you how the rich Calendar, AdRotator, MultiView, and Wizard controls can go far beyond the limitations of ordinary HTML elements.
- When you're working with these controls, you don't need to think about HTML at all. Instead, you can focus on the object model that's defined by the control.

8.5 Keywords

AdRotator Class: The actual AdRotator class provides a limited set of properties. You specify both the appropriate advertisement file in the AdvertisementFile property and the type of window that the link should follow

AdRotator: AdRotator is to provide a graphic on a page that is chosen randomly from a group of possible images.

CellPadding: CellPadding is the space, in pixels, between the border of each cell and its contents.

Keyword: A keyword that identifies a group of advertisements. You can use this for filtering.

MultiView: MultiView gives you a way to declare multiple views and show only one at a time.

8.6 Self Assessment

Fill in the blanks:

1. ..... property to configure how a week is selected.
2. The ..... provides a whole host of formatting-related properties.
3. ..... event occurs when the Calendar control is about to create a month to display to the user.
4. The Calendar control provides two useful events: SelectionChanged and .....
5. The basic purpose of the ..... is to provide a graphic on a page that is chosen randomly from a group of possible images.
6. The ..... is the simpler of the two multiple-view controls.

State whether the following statements are True or False:

7. The DayRender event is extremely powerful.
8. The actual AdRotator class do not provides a limited set of properties.
9. Each panel can hold a group of ASP.NET controls.
10. Visual Studio shows all your views at design time.

8.7 Review Questions

1. Explain the use of Calendar.FirstDayOfWeek property.
2. Describe the various properties of calendar styles.
3. What do you mean by AdRotator?
4. Explain AdRotator class in detail.

Notes

5. How will you create views? Explain with the help suitable example.
6. What do you mean by wizard control?
7. How will you formatting the calendar?
8. Describe the use of "SetActiveView()" function in views.
9. Describe CalenderDay properties.
10. Where you can use "NextPrevFormat" member?

Answers: Self Assessment

1.	Calendar.FirstDayOfWeek	2.	Calendar control
3.	Calendar.DayRender	4.	VisibleMonthChanged
5.	AdRotator	6.	MultiView
7.	True	8.	False
9.	True	10.	True

8.8 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Notes

Unit 9: User Controls and Graphics

CONTENTS

Objectives

Introduction

9.1 User Controls

9.1.1 Creating a Simple User Control

9.1.2 Independent User Controls

9.1.3 Integrated User Controls

9.2 User Control Events

9.3 Dynamic Graphics

9.3.1 Basic Drawing

9.3.2 Drawing a Custom Image

9.3.3 Placing Custom Images Inside Web Pages

9.3.4 Image Format and Quality

9.4 Summary

9.5 Keywords

9.6 Self Assessment

9.7 Review Questions

9.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Define user controls
- Describe dynamic graphics

Introduction

In this unit, you'll consider two ways to extend your web pages another notch.

First, you'll tackle user controls, which give you an efficient way to reuse a block of user interface markup and the code that goes with it. User controls are a key tool for building modular web applications. They can also help you create consistent website designs and reuse your hard work.

Next, you'll explore custom drawing with GDI+. You'll see how you can paint exactly the image you need on request. You'll also learn the best way to incorporate these images into your web pages.

Notes

9.1 User Controls

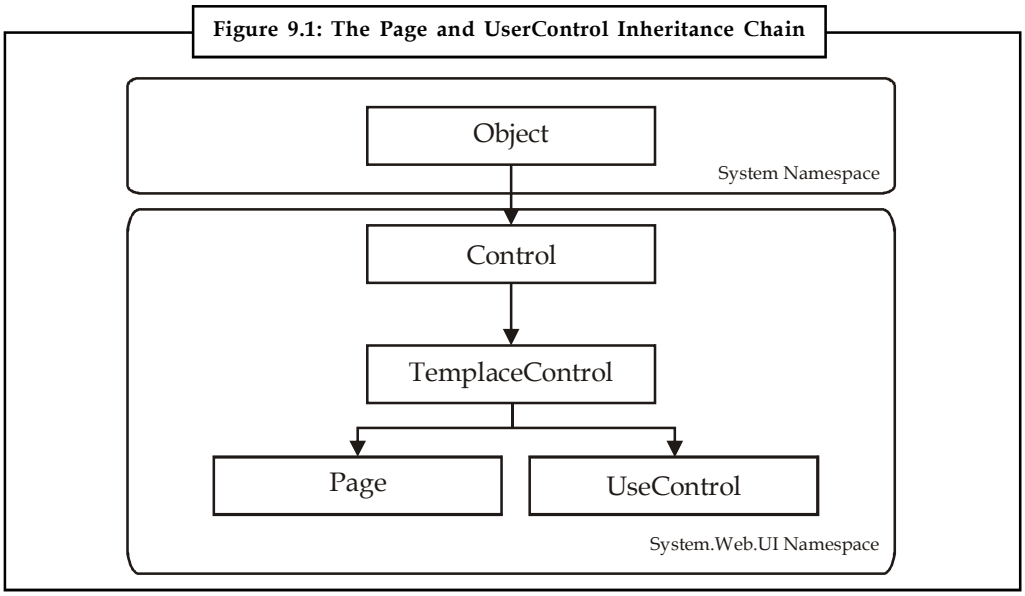
A well-built web application divides its work into discrete, independent blocks. The more modular your web application is, the easier it is to maintain your code, troubleshoot problems, and reuse key bits of functionality.

Although it's easy enough to reuse code (you simply need to pull it out of your pages and put it into separate classes), it's not as straightforward to reuse web page markup. You can cut and paste blocks of HTML and ASP.NET control tags, but this causes endless headaches if you want to change your markup later. Instead, you need a way to wrap up web page markup in a reusable package, just as you can wrap up ordinary C# code. The trick is to create a user control.

User controls look pretty much the same as ASP.NET web forms. Like web forms, they are composed of a markup portion with HTML and control tags (the .ascx file) and can optionally use a code-behind file with event-handling logic. They can also include the same range of HTML content and ASP.NET controls, and they experience the same events as the Page object (such as Load and PreRender). The only differences between user controls and web pages are as follows:

1. User controls use the file extension .ascx instead of .aspx, and their code-behind files inherit from the System.Web.UI.UserControl class. In fact, the UserControl class and the Page class both inherit from the same base classes, which is why they share so many of the same methods and events, as shown in the inheritance diagram in Figure 9.1.
2. The .ascx file for a user control begins with a <%@ Control %> directive instead of a <%@ Page %> directive.
3. User controls can't be requested directly by a web browser. Instead, they must be embedded inside other web pages.

Figure 9.1: The Page and UserControl Inheritance Chain



9.1.1 Creating a Simple User Control

You can create a user control in Visual Studio in much the same way you add a web page. Just select Website → Add New Item, and choose Web User Control from the list.

Notes

The following user control contains a single Label control:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="Footer.ascx.cs" Inherits="Footer" %>

<asp:Label id="lblFooter" runat="server" />
```

Note that the Control directive uses the same attributes used in the Page directive for a web page, including Language, AutoEventWireup, and Inherits.

The code-behind class for this sample user control is similarly straightforward. It uses the UserControl.Load event to add some text to the label:

```
public partial class Footer : System.Web.UI.UserControl
{
protected void Page_Load(Object sender, EventArgs e)
{
lblFooter.Text = "This page was served at ";

lblFooter.Text += DateTime.Now.ToString();
}
}
```

To test this user control, you need to insert it into a web page. This is a two-step process.

First, you need to add a Register directive to the page that will contain the user control. You place the Register directive immediately after the Page directive. The Register directive identifies the control you want to use and associates it with a unique control prefix, as shown here:

```
<%@ Register TagPrefix="apress" TagName="Footer" Src="Footer.ascx" %>
```

The Register directive specifies a tag prefix and name. Tag prefixes group sets of related controls (for example, all ASP.NET web controls use the tag prefix asp). Tag prefixes are usually lowercase technically, they are case-insensitive and should be unique for your company or organization. The Src directive identifies the location of the user control template file, not the code-behind file.

Second, you can now add the user control whenever you want (and as many times as you want) in the page by inserting its control tag. Consider this page example:

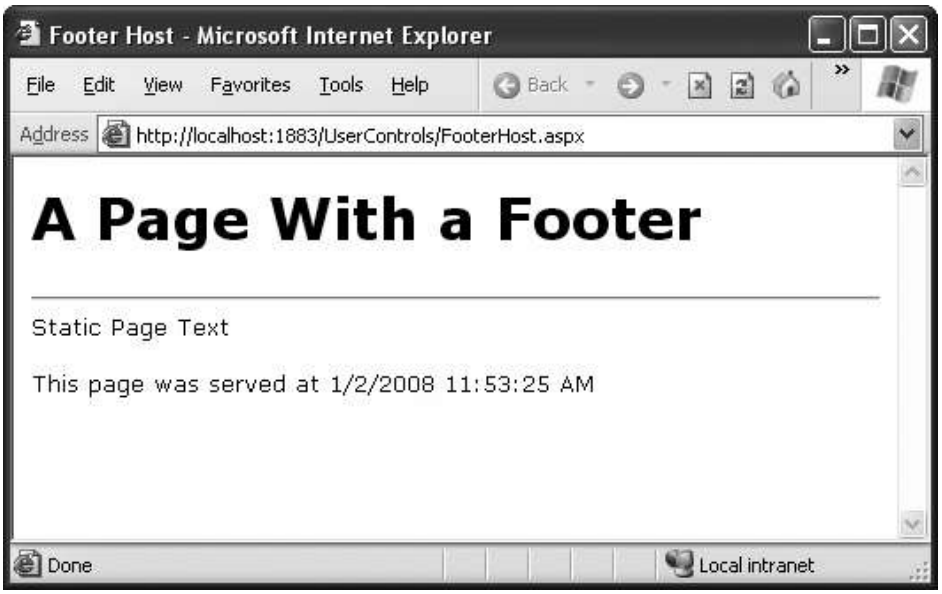
```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="FooterHost.aspx.cs" Inherits="FooterHost"%>
<%@ Register TagPrefix="apress" TagName="Footer" Src="Footer.ascx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Footer Host</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<h1>A Page With a Footer</h1><hr />
Static Page Text<br /><br />
```

Notes

```
<apress:Footer id="Footer1" runat="server" />
</div>
</form>
</body>
</html>
```

This example (Figure 9.2) demonstrates a simple way that you can create a header or footer and reuse it in all the pages in your website just by adding a user control. In the case of your simple footer, you won't save much code. However, this approach will become much more useful for a complex control with extensive formatting or several contained controls.

Figure 9.2: A Page with a User Control Footer



Of course, this only scratches the surface of what you can do with a user control. In the following sections, you'll learn how to enhance a control with properties, methods, and events transforming it from a simple "include file" into a full-fledged object.

Notes The Page class provides a special LoadControl() method that allows you to create a user control dynamically at runtime from an .ascx file. The user control is returned to you as a control object, which you can then add to the Controls collection of a container control on the web page (such as PlaceHolder or Panel) to display it on the page. This technique isn't a good substitute for declaratively using a user control, because it's more complex. However, it does have some interesting applications if you want to generate a user interface dynamically.

In Visual Studio, you have a useful shortcut for adding a user control to a page without typing the Register directive by hand. Start by opening the web page you want to use. Then, find the .ascx file for the user control in the Solution Explorer. Drag it from the Solution Explorer and drop it onto the visual design area of your web form (not the source view area). Visual Studio will automatically add the Register directive for the user control, as well as an instance of the user control tag.

Notes

9.1.2 Independent User Controls

Conceptually, two types of user controls exist: independent and integrated. Independent user controls don't interact with the rest of the code on your form. The Footer user control is one such example. Another example might be a LinkMenu control that contains a list of buttons offering links to other pages. This LinkMenu user control can handle the events for all the buttons and then run the appropriate Response.Redirect() code to move to another web page. Or it can just be an ordinary HyperLink control that doesn't have any associated server-side code. Every page in the website can then include the same LinkMenu user control, enabling painless website navigation with no need to worry about frames.

Notes You can use the more feature-rich navigation controls to provide website navigation. Creating your own custom controls gives you a simple, more flexible, but less powerful approach to providing navigation. You might use custom controls rather than a whole site map for straightforward navigation between a few pages.

The following sample defines a simple control that presents an attractively formatted list of links. Note that the style attribute of the tag (which defines fonts and formatting) has been omitted for clarity.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="LinkMenu.ascx.cs" Inherits="LinkMenu" %>
<div>
Products:<br />
<asp:HyperLink id="lnkBooks" runat="server"
NavigateUrl="MenuHost.aspx?product=Books">Books
</asp:HyperLink><br />
<asp:HyperLink id="lnkToys" runat="server"
NavigateUrl="MenuHost.aspx?product=Toys">Toys
</asp:HyperLink><br />
<asp:HyperLink id="lnkSports" runat="server"
NavigateUrl="MenuHost.aspx?product=Sports">Sports
</asp:HyperLink><br />
<asp:HyperLink id="lnkFurniture" runat="server"
NavigateUrl="MenuHost.aspx?product=Furniture">Furniture
</asp:HyperLink>
</div>
```

The links don't actually trigger any server-side code instead, they render themselves as ordinary HTML anchor tags with a hard-coded URL.

To test this menu, you can use the following MenuHost.aspx web page. It includes two controls: the Menu control and a Label control that displays the product query string parameter. Both are positioned using a table.

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="MenuHost.aspx.cs" Inherits="MenuHost"%>
<%@ Register TagPrefix="apress" TagName="LinkMenu" Src="LinkMenu.ascx" %>
```

Notes

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Menu Host</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<table>
<tr>
<td><apress:LinkMenu id="Menu1" runat="server" /></td>
<td><asp:Label id="lblSelection" runat="server" /></td>
</tr>
</table>
</div>
</form>
</body>
</html>
```

When the MenuHost.aspx page loads, it adds the appropriate information to the lblSelection control:

```
protected void Page_Load(Object sender, EventArgs e)
{
if (Request.Params["product"] != null)
{
lblSelection.Text = "You chose: ";
lblSelection.Text += Request.Params["product"];
}
}
```

Figure 9.3 shows the end result. Whenever you click a button, the page is posted back, and the text is updated.

Figure 9.3: The LinkMenu User Control

Notes

You could use the LinkMenu control to repeat the same menu on several pages. This is particularly handy in a situation where you can't use master pages to standardize layout (possibly because the pages are too different).

Task

Experiment with VBScript code that could imitate the functionality of the MARQUEE tag.

9.1.3 Integrated User Controls

Integrated user controls interact in one way or another with the web page that hosts them.

A typical example is a user control that allows some level of configuration through properties. For instance, you can create a footer that supports two different display formats: long date and short time. To add a further level of refinement, the Footer user control allows the web page to specify the appropriate display format using an enumeration.

The first step is to create an enumeration in the custom Footer class. Remember, an enumeration is simply a type of constant that is internally stored as an integer but is set in code by using one of the allowed names you specify. Variables that use the FooterFormat enumeration can take the value FooterFormat.LongDate or FooterFormat.ShortTime:

```
public enum FooterFormat
{
LongDate,
ShortTime
}
```

The next step is to add a property to the Footer class that allows the web page to retrieve or set the current format applied to the footer. The actual format is stored in a private variable called _format, which is set to the long date format by default when the control is first created. (You can accomplish the same effect, in a slightly sloppier way, by using a public member variable named Format instead of a full property procedure.)

```
private FooterFormat format = FooterFormat.LongDate;

public FooterFormat Format
{
get { return format; }
set { format = value; }
}
```

Finally, the UserControl.Load event handler needs to take account of the current footer state and format the output accordingly. The following is the full Footer class code:

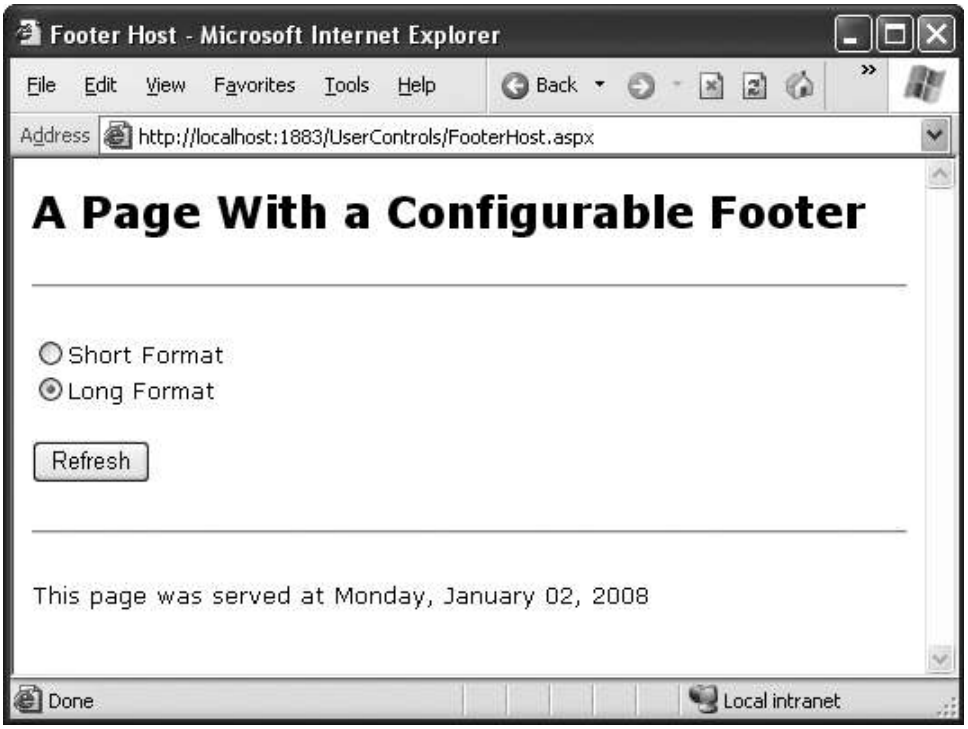
```
public partial class Footer : System.Web.UI.UserControl
{
public enum FooterFormat
{ LongDate, ShortTime }
private FooterFormat format = FooterFormat.LongDate;
public FooterFormat Format
{
get
```

Notes

```
{ return format; }
set
{ format = value; }
}
protected void Page_Load(Object sender, EventArgs e)
{
lblFooter.Text = "This page was served at ";
if (format == FooterFormat.LongDate)
{
lblFooter.Text += DateTime.Now.ToLongDateString();
}
else if (format == FooterFormat.ShortTime)
{
lblFooter.Text += DateTime.Now.ToShortTimeString();
}
}
}
```

To test this footer, you need to create a page that modifies the Format property of the Footer user control. Figure 9.4 shows an example page, which automatically sets the Format property for the user control to match a radio button selection whenever the page is posted back.

Figure 9.4: The Modified Footer



Note that the user control property is modified in the Page.Load event handler, not the cmdRefresh.Click event handler. The reason is that the Load event occurs before the user control has been rendered each time the page is created. The Click event occurs after the user control has

Notes

been rendered, and though the property change is visible in your code, it doesn't affect the user control's HTML output, which has already been added to the page.

```
public partial class FooterHost : System.Web.UI.Page
{
protected void Page_Load(Object sender, EventArgs e)
{
if (optLong.Checked)
{
Footer1.Format = Footer.FooterFormat.LongDate;
}
else if (optShort.Checked)
{
Footer1.Format = Footer.FooterFormat.ShortTime;
}
else
{
// The default value in the Footer class will apply.
}
}
}
```

You can also set the initial appearance of the footer in the control tag:

```
<apress:Footer Format="ShortTime" id="Footer1" runat="server" />
```

9.2 User Control Events

Another way that communication can occur between a user control and a web page is through events.With methods and properties, the user control reacts to a change made by the web page code. With events, the story is reversed: the user control notifies the web page about an action, and the web page code responds.

Creating a web control that uses events is fairly easy. In the following example, you'll see a version of the LinkMenu control that uses events. Instead of navigating directly to the appropriate page when the user clicks a button, the control raises an event, which the web page can choose to handle.

The first step to create this control is to define the events. Remember, to define an event, you must first choose an event signature. The .NET standard for events specifies that every event should use two parameters. The first one provides a reference to the control that sent the event, while the second incorporates any additional information. This additional information is wrapped into a custom EventArgs object, which inherits from the System.EventArgs class. (If your event doesn't require any additional information, you can just use the predefined EventArgs class, which doesn't contain any additional data. Many events in ASP.NET, such as Page.Load or Button.Click, follow this pattern.)

The LinkMenu2 control uses a single event, which indicates when a link is clicked:

```
public partial class LinkMenu2 : System.Web.UI.UserControl
{
public event EventHandler LinkClicked;
...
}
```

Notes

This code defines an event named LinkClicked. The LinkClicked event has the signature specified by the System.EventHandler delegate, which includes two parameters the event sender and an ordinary EventArgs object. That means that any event handler you create to handle the LinkClicked event must look like this:

```
protected void LinkMenu_LinkClicked(object sender, EventArgs e)
{ ... }
```

This takes care of defining the event, but what about raising it? This part is easy. To fire the event, the LinkMenu2 control simply calls the event by name and passes in the two parameters, like this:

```
// Raise the LinkClicked event, passing a reference to
// the current object (the sender) and an empty EventArgs object.
LinkClicked(this, EventArgs.Empty);
```

The LinkMenu2 control actually needs a few more changes. The original version used the HyperLink control. This won't do, because the HyperLink control doesn't fire an event when the link is clicked. Instead, you'll need to use the LinkButton. The LinkButton fires the Click event, which the LinkMenu2 control can intercept, and then raises the LinkClicked event to the web page.

The following is the full user control code:

```
public partial class LinkMenu2 : System.Web.UI.UserControl
{
public event EventHandler LinkClicked;
protected void lnk_Click(object sender, EventArgs e)
{
// One of the LinkButton controls has been clicked.
// Raise an event to the page.
if (LinkClicked != null)
{
LinkClicked(this, EventArgs.Empty);
}
}
}
```

Notice that before raising the LinkClicked event, the LinkMenu2 control needs to test the LickedClick event for a null reference. A null reference exists if no event handlers are attached to the event. In this case, you shouldn't try to raise the event, because it would only cause an error.

You can create a page that uses the LinkMenu2 control and add an event handler. Unfortunately, you won't be able to connect these event handlers using the Visual Studio Properties window, because the Properties window won't show the custom events that the user control provides. Instead, you'll need to modify the LinkMenu2 tag directly, as shown here:

```
<apress:LinkMenu2 id="Menu1" runat="server" OnLinkClicked="LinkClicked" /
>
```

and here's the event handler that responds in the web page:

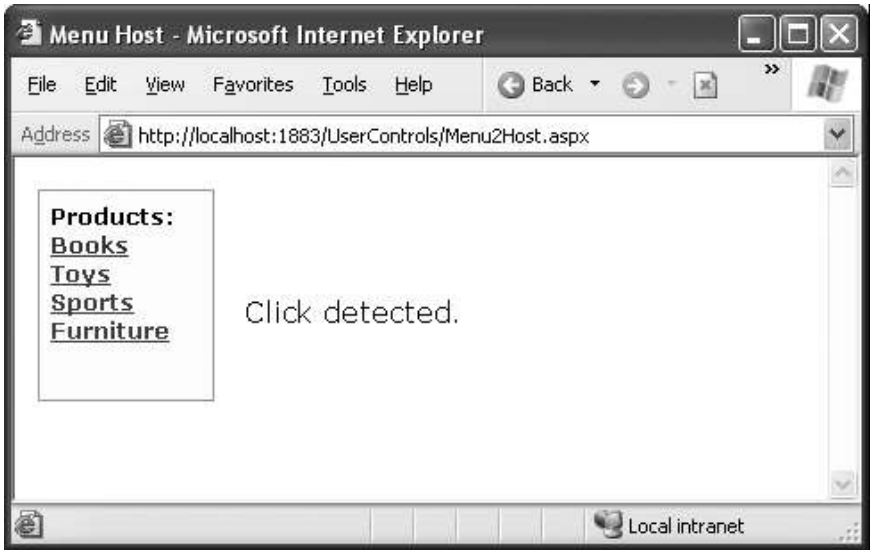
```
protected void LinkClicked(object sender, EventArgs e)
{
```

Notes

```
lblClick.Text = "Click detected.";
}
```

Figure 9.5 shows the result.

Figure 9.5: Using the LinkMenu2 User Control



Conceptually, this approach should give your web page more power to customize how the user control works. Unfortunately, that's not the case at the moment, because a key piece of information is missing. When the LinkClicked event occurs, the web page has no way of knowing what link was clicked, which prevents it from taking any kind of reasonable action. The only way to solve this problem is to create a more intelligent event that can transmit some information through event arguments.

Passing Information with Events

In the current LinkMenu2 example no custom information is passed along with the event. In many cases, however, you want to convey additional information that relates to the event. To do so, you need to create a custom class that derives from EventArgs.

The LinkClickedEventArgs class that follows allows the LinkMenu2 user control to pass the URL that the user selected through a Url property. It also provides a Cancel property. If set to true, the user control will stop its processing immediately. But if Cancel remains false (the default), the user control will send the user to the new page. This way, the user control still handles the task of redirecting the user, but it allows the web page to plug into this process and change it or stop it (for example, if there's unfinished work left on the current page).

```
public class LinkClickedEventArgs : EventArgs
{
private string url;
public string Url
{
get { return url; }
set { url = value; }
}
```

Notes

```
private bool cancel = false;
public bool Cancel
{
get { return cancel; }
set { cancel = value; }
}
public LinkClickedEventArgs(string url)
{
Url = url;
}
}
```

To use this EventArgs class, you need to create a new delegate that represents the LinkClicked event signature. Here's what it looks like:

```
public delegate void LinkClickedEventHandler(object sender,
LinkClickedEventArgs e);
```

Both the LinkClickedEventArgs class and the LinkClickedEventHandler delegate should be placed in the App_Code directory. That way, these classes will be compiled automatically and made available to all web pages.

Now you can modify the LinkClicked event to use the LinkClickedEventHandler delegate:

```
public event LinkClickedEventHandler LinkClicked;
```

Next, your user control code for raising the event needs to submit the required information when calling the event. But how does the user control determine what link was clicked? The trick is to switch from the LinkButton.Click event to the LinkButton.Command event. The Command event automatically gets the CommandArgument that's defined in the tag. So if you define your LinkButton controls like this:

```
<asp:LinkButton ID="lnkBooks" runat="server"
CommandArgument="Menu2Host.aspx?product=Books" OnCommand="lnk_Command">Books
</asp:LinkButton><br />
<asp:LinkButton ID="lnkToys" runat="server"
CommandArgument="Menu2Host.aspx?product=Toys" OnCommand="lnk_Command">Toys
</asp:LinkButton><br />
<asp:LinkButton ID="lnkSports" runat="server"
CommandArgument="Menu2Host.aspx?product=Sports"
OnCommand="lnk_Command">Sports
</asp:LinkButton><br />
<asp:LinkButton ID="lnkFurniture" runat="server"
CommandArgument="Menu2Host.aspx?product=Furniture" OnCommand="lnk_Command">
Furniture</asp:LinkButton>
```

You can pass the link along to the web page like this:

```
LinkClickedEventArgs args = new
LinkClickedEventArgs((string)e.CommandArgument);
LinkClicked(this, args);
```

Here's the complete user control code. It implements one more feature. After the event has been raised and handled by the web page, the LinkMenu2 checks the Cancel property. If it's false, it goes ahead and performs the redirect using Reponse.Redirect().

Notes

```
public partial class LinkMenu2 : System.Web.UI.UserControl
{
public event LinkClickedEventHandler LinkClicked;
protected void lnk_Command(object sender, CommandEventArgs e)
{
// One of the LinkButton controls has been clicked.
// Raise an event to the page.
if (LinkClicked != null)
{
// Pass along the link information.
LinkClickedEventArgs args =
new LinkClickedEventArgs((string)e.CommandArgument);
LinkClicked(this, args);
// Perform the redirect.
if (!args.Cancel)
{
// Notice we use the Url from the LinkClickedEventArgs
// object, not the original link. That means the web page
// can change the link if desired before the redirect.
Response.Redirect(args.Url);
}
}
}
}
```

Finally, you need to update the code in the web page (where the user control is placed) so that its event handler uses the new signature. In the following code, the LinkClicked event handler checks the URL and allows it in all cases except one:

```
protected void LinkClicked(object sender, LinkClickedEventArgs e)
{
if (e.Url == "Menu2Host.aspx?product=Furniture")
{
lblClick.Text = "This link is not allowed.";
e.Cancel = true;
}
else
{
// Allow the redirect, and don't make any changes to the URL.
}
}
```

If you click the Furniture link, you'll see the message shown in Figure 9.6.

Notes

Figure 9.6: Handling a User Control Event in the Page

9.3 Dynamic Graphics

One of the features of the .NET Framework is GDI+, a set of classes designed for drawing images. You can use GDI+ in a Windows or an ASP.NET application to create dynamic graphics. In a Windows application, the graphics you draw would be copied to a window for display. In ASP.NET, the graphics can be rendered right into the HTML stream and sent directly to the client browser.

In general, using GDI+ code to draw a graphic is slower than using a ready-made image file. However, GDI+ gives you much more freedom. For example, you can tailor your image to suit a particular purpose, incorporating information such as the date or current user name. You can also mingle text, shapes, and other bitmaps to create a complete picture.

9.3.1 Basic Drawing

You need to follow four basic steps when using GDI+. First, you have to create an in-memory bitmap. This is the drawing space where you'll create your masterpiece. To create the bitmap, declare a new instance of the System.Drawing.Bitmap class. You must specify the height and width of the image in pixels. Be careful don't make the bitmap larger than required, or you'll needlessly waste memory.

```
// Create an in-memory bitmap where you will draw the image.
// The Bitmap is 300 pixels wide and 50 pixels high.
Bitmap image = new Bitmap(300, 50);
```

The next step is to create a GDI+ graphics context for the image, which is represented by a System.Drawing.Graphics object. This object provides the methods that allow you to render content to the in-memory bitmap. To create a Graphics object from an existing Bitmap object, you just use the static Graphics.FromImage() method, as shown here:

```
Graphics g = Graphics.FromImage(image);
```

Notes

Notes The Graphics.FromImage() method works with any Image object. Classes such as Bitmap derive from Image, so they work fine.

Now comes the interesting part. Using the methods of the Graphics class, you can draw text, shapes, and images on the bitmap. Table 9.1 lists some of the most fundamental methods of the Graphics class. The methods that begin with the word Draw draw outlines, while the methods that begin with the word Fill draw solid regions. The only exceptions are the DrawString() method, which draws filled-in text using a font you specify, and the methods for copying bitmap images, such as DrawIcon() and DrawImage().

Table 9.1: DrawingMethods of the Graphics Class

Method	Description
DrawArc()	Draws an arc representing a portion of an ellipse specified by a pair of coordinates, a width, and a height (or some other combination of information, if you use one of the overloaded versions of this method).
DrawBezier() and DrawBeziers()	Draws the infamous and attractive Bezier curve, which is defined by four control points.
DrawClosedCurve()	Draws a curve and then closes it off by connecting the end points.
DrawCurve()	Draws a curve (technically, a cardinal spline).
DrawEllipse()	Draws an ellipse defined by a bounding rectangle specified by a pair of coordinates, a height, and a width.
DrawIcon() and DrawIconUnstreched()	Draws the icon represented by an Icon object and (optionally) stretches it to fit a given rectangle.
DrawImage() and DrawImageUnscaled()	Draws the image represented by an Image-derived object (such as a Bitmap object) and (optionally) stretches it to fit a given rectangle.
DrawLine() and DrawLines()	Draws one or more lines. Each line connects the two points specified by a coordinate pair.
DrawPie()	Draws a "piece of pie" shape defined by an ellipse specified by a coordinate pair, a width, a height, and two radial lines.
DrawPolygon()	Draws a multisided polygon defined by an array of points.
DrawRectangle() and DrawRectangles()	Draws one or more ordinary rectangles. Each rectangle is defined by a starting coordinate pair, a width, and a height.
DrawString()	Draws a string of text in a given font.
DrawPath()	Draws a more complex shape that's defined by the Path object.
FillClosedCurve()	Draws a curve, closes it off by connecting the end points, and fills it.
FillEllipse()	Fills the interior of an ellipse.
FillPie()	Fills the interior of a "piece of pie" shape.
FillPolygon()	Fills the interior of a polygon.
FillRectangle() and FillRectangles()	Fills the interior of one or more rectangles.
DrawPath()	Fills the interior of a complex shape that's defined by the Path object.

When calling the Graphics class methods, you need to specify several parameters to indicate the pixel coordinates for what you want to draw. For example, when drawing a rectangle, you need to specify the location of the top-left corner and its width and height.

Example: Here's an example of how you might draw a solid rectangle in yellow:

```
// Draw a rectangle starting at location (0, 0)
// that is 300 pixels wide and 50 pixels high.
g.FillRectangle(Brushes.Yellow, 0, 0, 300, 50);
```

Notes

When measuring pixels, the point (0, 0) is the top-left corner of your image in (x, y) coordinates. The x coordinate increases as you go farther to the right, and the y coordinate increases as you go farther down. In the current example, the image is 300 pixels wide and 50 pixels high, which means the point (299, 49) is the bottom-right corner.

Notes This code performs its drawing on the in-memory Bitmap object created earlier. Until this image is rendered (a skill you'll pick up shortly), you won't actually see anything on the web page.

You'll also notice that you need to specify either a Brush or a Pen object when you draw most content. (Both of these classes are defined in the System.Drawing namespace, alongside the Graphics class.) Methods that draw shape outlines require a Pen, while methods that draw filled-in shapes require a Brush. You can create your own custom Pen and Brush objects, but .NET provides an easier solution with the Brushes and Pens classes. These classes expose static properties that provide various Brushes and Pens for different colors. For example, Brushes.Yellow returns a Brush object that fills shapes using a solid yellow color.

Once the image is complete, you can send it to the browser using the Image.Save() method. Conceptually, you "save" the image to the browser's response stream. It then gets sent to the client and displayed in the browser.

```
// Render the image to the HTML output stream.
image.Save(Response.OutputStream,
System.Drawing.Imaging.ImageFormat.Gif);
```

Finally, you should explicitly release your image and graphics context when you're finished, because both hold onto some unmanaged resources that might not be released right away if you don't:

```
g.Dispose();
image.Dispose();
```

Task Create a Web page that contains several label controls of a variety of colors and sizes and animate each of them.

9.3.2 Drawing a Custom Image

Using the techniques you've learned, it's easy to create a simple web page that uses GDI+. The next example uses GDI+ to render some text in a bordered rectangle with a happy-face graphic next to it.

Here's the code you'll need:

```
protected void Page_Load(Object sender, EventArgs e)
{
// Create an in-memory bitmap where you will draw the image.
// The Bitmap is 300 pixels wide and 50 pixels high.
Bitmap image = new Bitmap(300, 50);

// Get the graphics context for the bitmap.
Graphics g = Graphics.FromImage(image);
```

Notes

```
// Draw a solid yellow rectangle with a red border.
g.FillRectangle(Brushes.LightYellow, 0, 0, 300, 50);
g.DrawRectangle(Pens.Red, 0, 0, 299, 49);

// Draw some text using a fancy font.
Font font = new Font("Alba Super", 20, FontStyle.Regular);
g.DrawString("This is a test.", font, Brushes.Blue, 10, 0);

// Copy a smaller gif into the image from a file.
System.Drawing.Image icon = Image.FromFile(Server.MapPath("smiley.gif"));
g.DrawImageUnscaled(icon, 240, 0);

// Render the entire bitmap to the HTML output stream.
image.Save(Response.OutputStream,
System.Drawing.Imaging.ImageFormat.Gif);

// Clean up.
g.Dispose();
image.Dispose();
}
```

This code is easy to understand. It follows the basic pattern set out earlier it creates the in-memory Bitmap, gets the corresponding Graphics object, performs the painting, and then saves the image to the response stream. This example uses the FillRectangle(), DrawRectangle(), DrawString(), and DrawImageUnscaled() methods to create the complete drawing shown in Figure 9.7.

Figure 9.7: Drawing a Custom Image

9.3.3 Placing Custom Images Inside Web Pages

The Image.Save() approach demonstrated so far has one problem. When you save an image to the response stream, you overwrite whatever information ASP.NET would otherwise use. If

Notes

you have a web page that includes other content and controls, this content won't appear at all in the final web page. Instead, the dynamically rendered graphics replace it.

Fortunately, this has a simple solution: you can link to a dynamically generated image using the HTML <img> tag or the Image web control. But instead of linking your image to a fixed image file, link it to the .aspx file that generates the picture.

Example: You could create a file named GraphicalText.aspx that writes a dynamically generated image to the response stream. In another page, you could show the dynamic image by adding an Image web control and setting the ImageUrl property to GraphicalText.aspx. In fact, you'll even see the image appear in Visual Studio's design-time environment before you run the web page!

When you use this technique to embed dynamic graphics in web pages, you also need to think about how the web page can send information to the dynamic graphic. For example, what if you don't want to show a fixed piece of text, but instead you want to generate a dynamic label that incorporates the name of the current user? (In fact, if you do want to show a fixed piece of text, it's probably better to create the graphic ahead of time and store it in a file, rather than generating it using GDI+ code each time the user requests the page.). The page that renders the graphic can then check for the query string information it needs.

Here's how you'd rewrite the dynamic graphic generator with this in mind:

```
// Get the user name.
if (Request.QueryString["Name"] == null)
{
// No name was supplied.
// Don't display anything.
}
else
{
string name = Request.QueryString["Name"];
// Create an in-memory bitmap where you will draw the image.
Bitmap image = new Bitmap(300, 50);
// Get the graphics context for the bitmap.
Graphics g = Graphics.FromImage(image);
g.FillRectangle(Brushes.LightYellow, 0, 0, 300, 50);
g.DrawRectangle(Pens.Red, 0, 0, 299, 49);
// Draw some text based on the query string.
Font font = new Font("Alba Super", 20, FontStyle.Regular);
g.DrawString(name, font, Brushes.Blue, 10, 0);
// Render the entire bitmap to the HTML output stream.
image.Save(Response.OutputStream,
System.Drawing.Imaging.ImageFormat.Gif);
g.Dispose();
image.Dispose();
}
```

Conceptually, this code isn't much different than the examples you've seen before. The only change is that one piece of information the string that's used with the DrawString() method is retrieved from the query string.

Notes

Figure 9.8 shows a page that uses this dynamic graphic page, along with two Label controls. The page passes the query string argument Joe Brown to the page. The full Image.ImageUrl thus becomes GraphicalText.aspx?Name=Joe%20Brown, as shown here:

```
<asp:Label id="Label1" runat="server">Here is some content.</asp:Label>
<br /><br />
<asp:Image id="Image1" runat="server"
ImageUrl="GraphicalText2.aspx?Name=Joe%20Brown"></asp:Image>
<br /><br />
<asp:Label id="Label2" runat="server">Here is some more content.</asp:Label>
```

Figure 9.8: Mingling Custom Images and Controls on the same Page

It's possible that you might need to send more information or more complex information to the page that draws the image. For example, you might want to pass a data object to a page that draws a pie chart. In this case, the query string isn't good enough, and you'll need to use a different type of state management.

9.3.4 Image Format and Quality

When you render an image, you can also choose the format you want to use. JPEG offers the best color support and graphics, although it uses compression that can lose detail and make text look fuzzy. GIF (the standard used in the examples so far) is often a better choice for graphics containing text, but it doesn't offer good support for color. In .NET, every GIF uses a fixed palette with 256 generic colors. If you use a color that doesn't map to one of these presets, the color will be dithered, leading to a less-than-optimal graphic.

However, the best format choice is PNG. PNG is an all-purpose format that always provides high quality by combining the lossless compression of GIFs with the rich color support of JPEGs. Unfortunately, browsers such as Internet Explorer often don't handle it correctly when you return PNG content directly from a page. Instead of seeing the picture content, you'll receive a message prompting you to download the picture content and open it in another program. To sidestep this problem, you need to use the <img> tag approach shown in the previous example.

You need to be aware of two more quirks when using PNG. First, some older browsers (including Netscape 4.x) don't support PNG. Second, you can't use the Bitmap.Save() method shown in

Notes

earlier examples. If you do, an error will occur. (Technically, the problem is the Save() method requires a seekable stream a stream where the position can be changed at will. That's because .NET needs to be able to move back and forth through the picture content while it's being generated.)

The solution is easy to implement, if a little awkward. Instead of saving directly to Response.OutputStream, you can create a System.IO.MemoryStream object, which represents an in-memory buffer of data. The MemoryStream is always seekable, so you can save the image to this object. Once you've performed this step, you can easily copy the data from the MemoryStream to the Response.OutputStream. The only disadvantage is that this technique requires more memory because the complete rendered content of the graphic needs to be held in memory at once. However, the graphics you use in web pages generally aren't that large, so you probably won't observe any reduction in performance.

To implement this solution, start by importing the System.IO namespace:

```
using System.IO;
```

Now you can replace the previous example with this modified code that saves the image in PNG format. The changed lines are highlighted.

```
// Get the user name.
if (Request.QueryString["Name"] == null)
{
// No name was supplied.
// Don't display anything.
}
else
{
string name = Request.QueryString["Name"];

// Create an in-memory bitmap where you will draw the image.
Bitmap image = new Bitmap(300, 50);

// Get the graphics context for the bitmap.
Graphics g = Graphics.FromImage(image);
g.FillRectangle(Brushes.LightYellow, 0, 0, 300, 50);
g.DrawRectangle(Pens.Red, 0, 0, 299, 49);

// Draw some text based on the query string.
Font font = new Font("Alba Super", 20, FontStyle.Regular);
g.DrawString(name, font, Brushes.Blue, 10, 0);
Response.ContentType = "image/png";

// Create the PNG in memory.
MemoryStream mem = new MemoryStream();
image.Save(mem, System.Drawing.Imaging.ImageFormat.Png);

// Write the MemoryStream data to the output stream.
mem.WriteTo(Response.OutputStream);
g.Dispose();
```

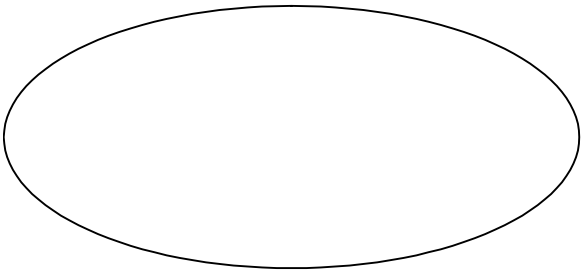
Notes

```
image.Dispose();
}
```

Quality isn't just determined by the image format. It also depends on the way you draw the image content onto the in-memory bitmap. GDI+ allows you to choose between optimizing your drawing code for appearance or speed. When you choose to optimize for the best appearance, .NET uses extra rendering techniques such as antialiasing to improve the drawing.

Antialiasing smooths jagged edges in shapes and text. It works by adding shading at the border of an edge. For example, gray shading might be added to the edge of a black curve to make a corner look smoother. Technically, antialiasing blends a curve with its background. Figure 9.9 shows a close-up of an antialiased ellipse.

Figure 9.9: Antialiasing with an Ellipse



To use smoothing in your applications, you set the SmoothingMode property of the Graphics object. You can choose between None, HighSpeed (the default), AntiAlias, and HighQuality (which is similar to AntiAlias but uses other, slower optimizations that improve the display on LCD screens). The Graphics.SmoothingMode property is one of the few stateful Graphics class members. This means you set it before you begin drawing, and it applies to any text or shapes you draw in the rest of the paint session (until the Graphics object is released).

```
g.SmoothingMode = Drawing.Drawing2D.SmoothingMode.AntiAlias;
```

You can also use antialiasing with fonts to soften jagged edges on text. You can set the Graphics.TextRenderingHint property to ensure optimized text. You can choose between SingleBitPerPixelGridFit (fastest performance and lowest quality), AntiAliasGridFit (better quality but slower performance), and ClearTypeGridFit (the best quality on an LCD display). Or you can use the SystemDefault value to apply whatever font-smoothing settings the user has configured. SystemDefault is the default setting, and the default system settings for most computers enable text antialiasing. Even if you don't set this, your dynamically rendered text will probably be drawn in high quality. However, because you can't necessarily control the system settings of the web server, it's a good practice to specify this setting explicitly if you need to draw text in an image.

Case Study

How to do File Upload with ASP.NET MVC?

We have to use the HttpPostedFileBase class, rather than the HttpPostedFile because every Request, Response, HttpContext and all related ASP.NET intrinsic abstractions are one layer farther way in ASP.NET MVC. If you get an HttpRequest in ASP.NET, then in ASP.NET MVC at runtime...

Contd...

Notes

You'll get an HttpRequestWrapper while running under a Webserver

You'll get a dynamically generated derived Mock of an HttpRequestBase while running outside a Webserver (like inside a test) when you've made your own ControllerContext.

In each case, the instances you'll get are both (ultimately) of type HttpRequestBase, but it's this extra layer of abstraction that makes ASP.NET MVC easy to test and ASP.NET WebForms less so. I hope these Wrappers will be included in a future release of WebForms. The fact that they live in the System.Web.Abstractions.dll and not System.Web.Mvc. Abstractions. dll tells me someone has their eye on that particular ball.

At any rate, here's the Controller that takes File Upload requests:

```
public class ViewDataUploadFilesResult
{
 public string Name { get; set; }
 public int Length { get; set; }
}

public class HomeController : Controller
{
 public ActionResult UploadFiles()
  {
 var r = new List<ViewDataUploadFilesResult>();

 foreach (string file in Request.Files)
  {
 HttpPostedFileBase hpf = Request.Files[file] as HttpPostedFileBase;
 if (hpf.ContentLength == 0)
 continue;
 string savedFileName = Path.Combine(
 AppDomain.CurrentDomain.BaseDirectory,
 Path.GetFileName(hpf.FileName));
 hpf.SaveAs(savedFileName);

 r.Add(new ViewDataUploadFilesResult()
 { Name = savedFileName,
 Length = hpf.ContentLength });
  }
 return View("UploadedFiles",r);
  }
}
```

At the bottom where I ask for the "UploadedFiles" view, and I pass in my list of ViewDataUploadFilesResults. This will appea in the ViewData.Model property. The View then displays them, and that's ALL the View does.

```
<ul>
<% foreach (ViewDataUploadFilesResult v in this.ViewData.Model) { %>
```

Contd...

Notes

```
<%=String.Format("<li>Uploaded: {0} totalling {1} bytes.</
li>",v.Name,v.Length) %>
<% } %>
</ul>
```

9.4 Summary

- In this unit, you put two more tools in your ASP.NET toolkit.
- First, you saw how user controls allow you to reuse a block of user interface in more than one web page.
- Next, you considered how custom drawing allows you to create made-to-measure graphics.

9.5 Keywords

DrawArc(): Draws an arc representing a portion of an ellipse specified by a pair of coordinates, a width, and a height (or some other combination of information, if you use one of the overloaded versions of this method).

GDI+: The portion of the Microsoft Windows XP operating system that provides two-dimensional vector graphics, imaging, and typography. GDI+ improves on GDI (the graphics device interface included with earlier versions of Windows) by adding new features and optimizing existing features. The GDI+ managed class interface (a set of wrappers) is part of the Microsoft .NET Framework.

Integrated User Controls: Integrated user controls interact in one way or another with the web page that hosts them.

User Control: In ASP.NET: A server control that is authored declaratively using the same syntax as an ASP.NET page and is saved as a text file with an .ascx extension. User controls allow page functionality to be partitioned and reused.

9.6 Self Assessment

Fill in the blanks:

1. A well-built web application divides its work into discrete .....
2. ..... look pretty much the same as ASP.NET web forms.
3. The Register directive specifies a tag prefix and .....
4. ..... user controls don't interact with the rest of the code on your form.
5. Using ..... code to draw a graphic is slower than using a ready-made image file.
6. To create the bitmap, declare a new instance of the ..... class.

State whether the following statements are True or False:

7. FillPolygon() method use to fills the interior of a polygon.
8. Antialiasing smooths jagged edges in shapes and text.
9. Creating a web control that uses events is fairly difficult.
10. The MemoryStream is always seekable.

Notes

9.7 Review Questions

1. What do you mean by user control?
2. Describe independent user control.
3. "Another way that communication can occur between a user control and a web page is through events." Explain.
4. How will you pass information with events?
5. What do you mean by GDI+?
6. Describe the drawing methods of graphic classes.
7. How will you draw a custom image? Explain.
8. How would you place a image into the web form? Explain.
9. What do you mean by integrated user control?
10. Describe the use of "Graphics.FromImage()" function.

Answers: Self Assessment

1.	independent blocks	2.	User controls
3.	name	4.	Independent
5.	GDI+	6.	System.Drawing.Bitmap
7.	True	8.	True
9.	False	10.	True

9.8 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Notes

Unit 10: Themes and Styles

CONTENTS

Objectives

Introduction

10.1 Applying a Theme to a Single ASP.NET Page

10.2 Applying a Theme to an entire Application

10.3 Removing Themes from Server Controls

10.4 Removing Themes from Web Pages

10.5 Creating your own Themes

10.5.1 Creating the Proper Folder Structure

10.5.2 Including CSS Files in your Themes

10.6 Defining Multiple Skin Options

10.7 Programmatically Working with Themes

10.8 Themes and Custom Controls

10.9 Summary

10.10 Keywords

10.11 Self Assessment

10.12 Review Questions

10.13 Further Readings

Objectives

After studying this unit, you will be able to:

- Define themes
- Describe styles

Introduction

When you build a Web application, it usually has a similar look-and-feel across all its pages. Not too many applications are designed with each page dramatically different from the next. Generally, for your applications, you use similar fonts, colors, and server control styles across all the pages.

You can apply these common styles individually to each and every server control or object on each page, or you can use a new capability provided by ASP.NET 2.0 to centrally specify these styles. All pages or parts of pages in the application can then access them.

Themes are the text-based style definitions in ASP.NET 2.0 that are the focus of this unit.

Notes

10.1 Applying a Theme to a Single ASP.NET Page

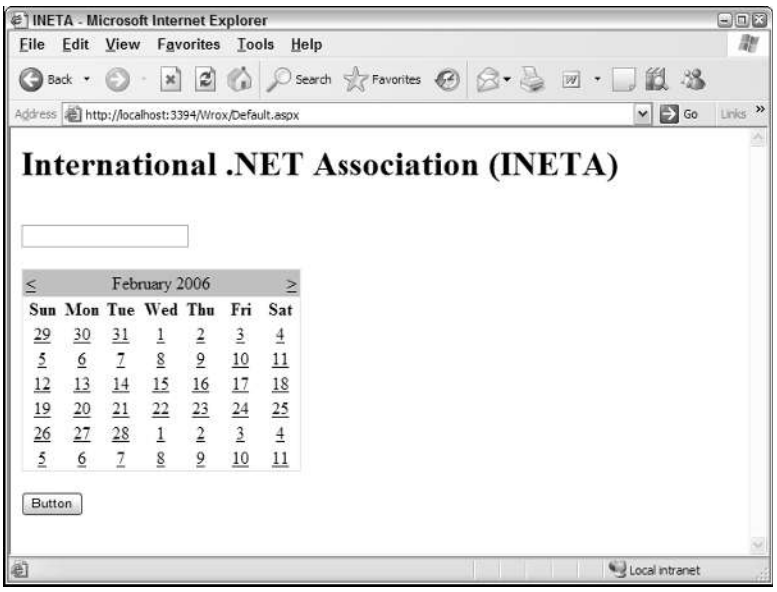
Themes are similar to Cascading Style Sheets (CSS) in that they enable you to define visual styles for your Web pages. Themes go further than CSS, however, in that they allow you to apply styles, graphics, and even CSS files themselves to the pages of your applications. You can apply ASP.NET themes at the application, page, or server control level.

In order to see how to use one of these themes, create a basic page, which includes some text, a text box, a button, and a calendar. This is shown in Listing 10.1.

Listing 10.1: An ASP.NET page that does not use themes

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

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>INETA</title>
</head>
<body>
<form id="form1" runat="server">
<h1>International .NET Association (INETA)</h1><br />
<asp:Textbox ID="TextBox1" Runat="server" /><br />
<br />
<asp:Calendar ID="Calendar1" Runat="server" /><br />
<asp:Button ID="Button1" Runat="server" Text="Button" />
</form>
</body>
</html>
```

This simple page shows some default server controls that appear just as you would expect, but that you can change with one of these new ASP.NET themes. When this theme-less page is called in the browser, it should look like figure.

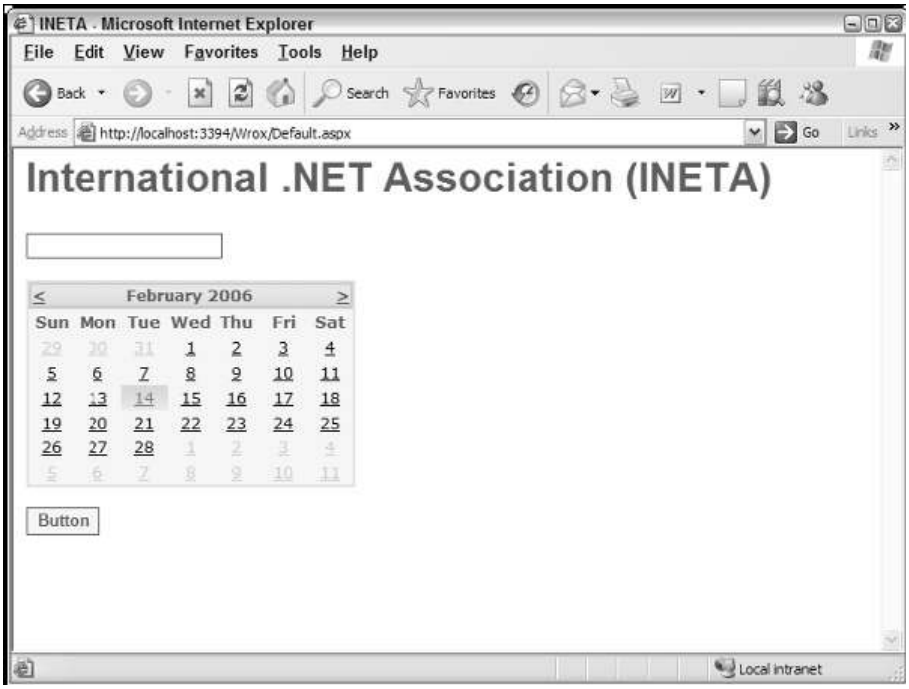


Notes

You can instantly change the appearance of this page without changing the style of each server control on the page. From within the Page directive, you simply apply an ASP.NET theme that you have either built (shown later in this unit) or downloaded from the Internet:

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

Adding the Theme attribute to the Page directive changes the appearance of everything on the page that is defined in an example SmokeAndGlass theme file. Using this theme, when I invoked the page in the browser, I got the result shown in figure.



From here, you can see that everything including the font, font color, text box, button, and more has changed appearance. If you have multiple pages, you may find that it's nice not to have to think about applying styles to everything you do as you build because the styles are already centrally defined for you.

10.2 Applying a Theme to an entire Application

In addition to applying an ASP.NET 2.0 theme to your ASP.NET pages using the Theme attribute within the Page directive, you can also apply it at an application level from the Web.config file. This is illustrated in Listing 10.2.

Listing 10.2: Applying a theme application-wide from the Web.config file

```
<?xml version="1.0" encoding="UTF-8" ?>

<configuration>
<system.web>
<pages theme="SmokeAndGlass" />
</system.web>
</configuration>
```

Notes

If you specify the theme in the Web.config file, you don't need to define the theme again in the Page directive of your ASP.NET pages. This theme is applied automatically to each and every page within your application.

10.3 Removing Themes from Server Controls

Whether themes are set at the application level or on a page, at times you want an alternative to the theme that has been defined.

Example: Change the text box server control that you have been working with (Listing 10.1) by making its background black and using white text:

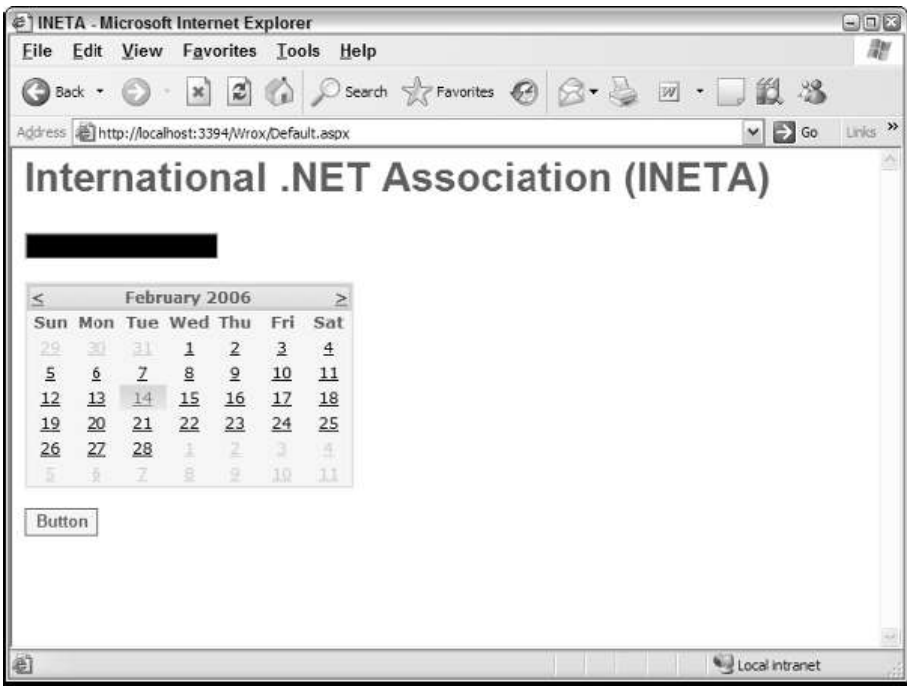
```
<asp:Textbox ID="TextBox1" Runat="server"
BackColor="#000000" ForeColor="#ffffff" />
```

The black background color and the color of the text in the text box are specified directly in the control itself with the use of the BackColor and ForeColor attributes. If you have applied a theme to the page where this text box control is located, however, you won't see this black background or white text because these changes are overridden by the theme itself.

To apply a theme to your ASP.NET page but not to this text box control, you simply use the EnableTheming property of the text box server control:

```
<asp:Textbox ID="TextBox1" Runat="server"
BackColor="#000000" ForeColor="#ffffff" EnableTheming="false" />
```

If you apply this property to the text box server control from Listing 10.1 while the SmokeAndGlass theme is still applied to the entire page, the theme is applied to every control on the page except the text box. This result is shown in figure.



Notes

If you want to turn off theming for multiple controls within a page, consider using the Panel control to encapsulate a collection of controls and then set the EnableTheming attribute of the Panel control to False. This disables theming for each control contained within the Panel control.

10.4 Removing Themes from Web Pages

Now what if, when you set the theme for an entire application in the Web.config file, you want to exclude a single ASP.NET page? It is quite possible to remove a theme setting at the page level, just as it is at the server control level.

The Page directive includes an EnableTheming attribute that can be used to remove theming from your ASP.NET pages. To remove the theme that would be applied by the theme setting in the Web.config, you simply construct your Page directive in the following manner:

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

This construct sets the theme to nothing thereby removing any settings that were specified in the Web.config file. When this directive is set to False at the page or control level, the Theme directory is not searched, and no .skin files are applied. When it is set to True at the page or control level, the Theme directory is searched and .skin files are applied.

If themes are disabled because the EnableTheming attribute is set to False at the page level, you can still enable theming for specific controls on this page by setting the EnableTheming property for the control to True and applying a specific theme at the same time, as illustrated here:

```
<asp:Textbox ID="TextBox1" Runat="server"
BackColor="#000000" ForeColor="#ffffff" EnableTheming="true" Theme="Summer
"/>
```

Understanding the StyleSheetTheme Attribute

The Page directive also includes the attribute StylesheetTheme that you can use to apply themes to a page. So, the big question is: If you have a Theme attribute and a StylesheetTheme attribute for the Page directive, what is the difference between the two?

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

The StylesheetTheme attribute works the same as the Theme attribute in that it can be used to apply a theme to a page. The difference is that the when attributes are set locally on the page within a particular control, the attributes are overridden by the theme if you use the Theme attribute. They are kept in place, however, if you apply the page's theme using the StylesheetTheme attribute. Suppose you have a text box control like the following:

```
<asp:Textbox ID="TextBox1" Runat="server"
BackColor="#000000" ForeColor="#ffffff" />
```

In this example, the BackColor and ForeColor settings are overridden by the theme if you have applied it using the Theme attribute in the Page directive. If, instead, you applied the theme using the StylesheetTheme attribute in the Page directive, the BackColor and ForeColor settings remain in place, even if they are explicitly defined in the theme.

10.5 Creating your own Themes

You will find that creating themes in ASP.NET is a rather simple process although sometimes it does require some artistic capabilities. The themes you create can be applied at the application, page, or server control level. Themes are a great way to easily apply a consistent look-and-feel across your entire application.

Notes

10.5.1 Creating the Proper Folder Structure

In order to create your own themes for an application, you first need to create the proper folder structure in your application. To do this, right-click your project and add a new folder. Name the folder App_Themes. You can also create this folder by right-clicking on your project in Visual Studio and selecting Add Folder'! Theme Folder. Notice when you do this that the App_Themes folder does not have the typical folder icon next to it, but instead has a folder icon that includes a paint brush. This is shown in figure.

Within the App_Themes folder, you create an additional theme folder for each and every theme that you might use in your application. For instance, if you are going to have four themes Summer, Fall, Winter, and Spring then you create four folders that are named appropriately.

You might use more than one theme in your application for many reasons season changes, day/night changes, different business units, category of user, or even user preferences.

Each theme folder must contain the elements of the theme, which can include the following:

1. A single skin file
2. CSS files
3. Images

Creating a Skin

A skin is a definition of styles applied to the server controls in your ASP.NET page. Skins can work in conjunction with CSS files or images. To create a theme to use in your ASP.NET applications, you use just a single skin file in the theme folder. The skin file can have any name, but it must have a .skin file extension.

Even though you have four theme folders in your application, concentrate on the creation of the Summer theme for the purposes of this unit. Right-click the Summer folder, select Add New

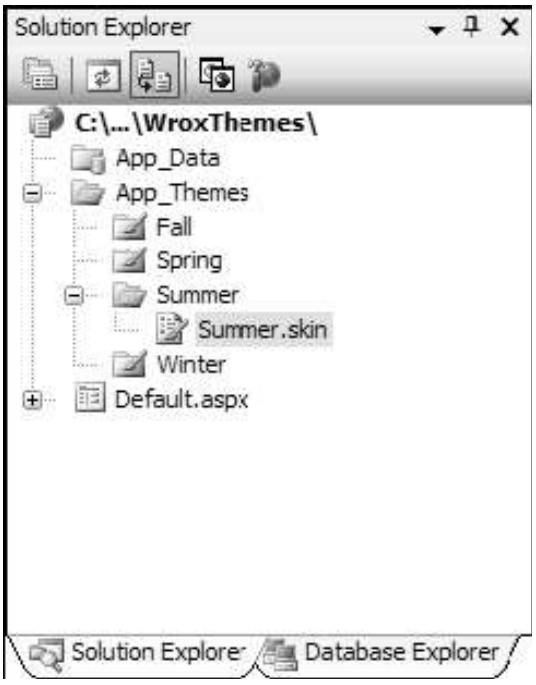
Notes

Item, and select Skin File from the listed options. Name the file Summer.skin. Then complete the skin file as shown in Listing 10.3.

Listing 10.3: The Summer.skin file

```
<asp:Label Runat="server" ForeColor="#004000" Font-Names="Verdana"
Font-Size="X-Small" />
<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"

Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
BorderColor="#004000" Font-Bold="True" />
<asp:Button Runat="server" ForeColor="#004000" Font-Names="Verdana"

Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
BorderColor="#004000" Font-Bold="True" BackColor="#FFE0C0" />
```

This is just a sampling of what the Summer.skin file should contain. To use it in a real application, you should actually make a definition for each and every server control option. In this case, you have a definition in place for three different types of server controls: Label, TextBox, and Button. After saving the Summer.skin file in the Summer folder, your file structure should resemble Figure shows the Solution Explorer of Visual Studio 2005.



Just like the regular server control definitions that you put on a typical .aspx page, these control definitions must contain the Runat="server" attribute. If you specify this attribute in the skinned version of the control, you also include it in the server control you put on an .aspx page that uses this theme. Also notice is that no ID attribute is specified in the skinned version of the control. If you specify an ID attribute here, you get an error when a page tries to use this theme.

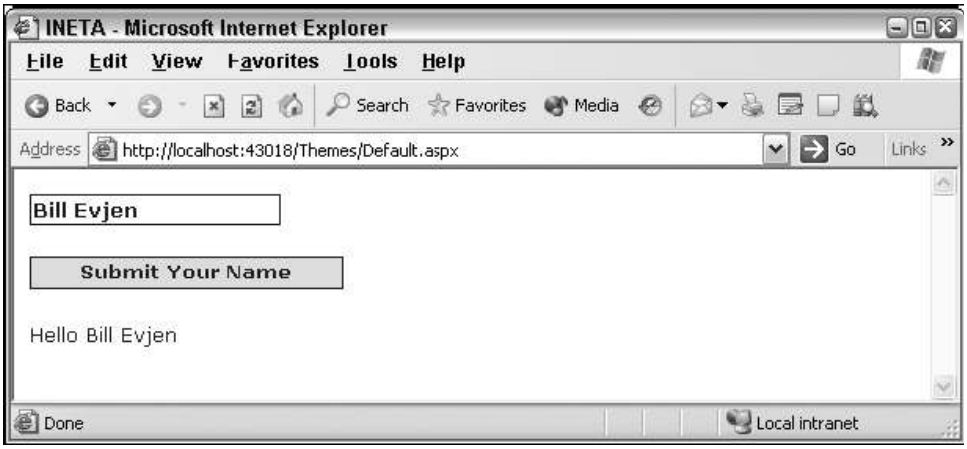
As you can see, you supply a lot of different visual definitions to these three controls, and this should give the page a summery look and feel. An ASP.NET page in this project can simply use this custom theme as was shown earlier in this unit (Listing 10.4).

Notes

Listing 10.4: Using the Summer theme in an ASP.NET page

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

<script runat="server">
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
Label1.Text = "Hello "& Textbox1.Text
End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>INETA</title>
</head>
<body>
<form id="form1" runat="server">
<asp:Textbox ID="TextBox1" Runat="server">
</asp:Textbox>
<br />
<br />
<asp:Button ID="Button1" Runat="server" Text="Submit Your Name"
OnClick="Button1_Click" />
<br />
<br />
<asp:Label ID="Label1" Runat="server" />
</form>
</body>
</html>
```

Looking at the server controls on this .aspx page, you can see that no styles are associated with them. These are just the default server controls that you drag and drop onto the design surface of Visual Studio 2005. There is, however, the style that you defined in the Summer.skin file, as shown in figure.



Notes

Task

Create a page with five toggle buttons and a check box control. When the check box is checked, only one button may remain down at any one time. When the check box is unchecked, all the buttons can be down at the same time.

10.5.2 Including CSS Files in your Themes

What is CSS?

A CSS (cascading style sheet) file allows you to separate your web sites (X)HTML content from it's style. As always you use your (X)HTML file to arrange the content, but all of the presentation (fonts, colors, background, borders, text formatting, link effects & so on…) are accomplished within a CSS.

At this point you have some choices of how to use the CSS, either internally or externally.

Internal Stylesheet

First we will explore the internal method. This way you are simply placing the CSS code within the <head></head> tags of each (X)HTML file you want to style with the CSS. The format for this is shown in the example below.

```
<head>
<title><title>
<style type="text/css">
CSS Content Goes Here
</style>
</head>
<body>
```

With this method each (X)HTML file contains the CSS code needed to style the page. Meaning that any changes you want to make to one page, will have to be made to all. This method can be good if you need to style only one page, or if you want different pages to have varying styles.

External Stylesheet

Next we will explore the external method. An external CSS file can be created with any text or HTML editor such as "Notepad" or "Dreamweaver". A CSS file contains no (X)HTML, only CSS. You simply save it with the .css file extension. You can link to the file externally by placing one of the following links in the head section of every (X)HTML file you want to style with the CSS file.

```
<link rel="stylesheet" type="text/css" href="Path To stylesheet.css" />
```

Or you can also use the @import method as shown below

```
<style type="text/css">@import url(Path To stylesheet.css)</style>
```

Notes

Either of these methods are achieved by placing one or the other in the head section as shown in example below.

```
<head>
<title><title>
<link rel="stylesheet" type="text/css"href="style.css" />
</head>
<body>
```

or

```
<head>
<title><title>
<style type="text/css"> @import url(Path To stylesheet.css) </style>
</head>
<body>
```

By using an external style sheet, all of your (X)HTML files link to one CSS file in order to style the pages. This means, that if you need to alter the design of all your pages, you only need to edit one .css file to make global changes to your entire website.

Here are a few reasons this is better.

Easier Maintenance

Reduced File Size

Reduced Bandwidth

Improved Flexibility

CSS Syntax

The syntax for CSS is different than that of (X)HTML markup. Though it is not too confusing, once you take a look at it. It consists of only 3 parts.

selector { property: value }

The selector is the (X)HTML element that you want to style. The property is the actual property title, and the value is the style you apply to that property.

Each selector can have multiple properties, and each property within that selector can have independent values. The property and value are separated with a colon and contained within curly brackets. Multiple properties are separated by a semi colon. Multiple values within a property are sperated by commas, and if an individual value contains more than one word you surround it with quotation marks. As shown below.

```
body {
  background: #eeeeee;
  font-family: "Trebuchet MS", Verdana, Arial, serif;
}
```

As you can see in the above code I have separated the color from the font-family with a semi-colon, separated the various fonts with commas and contained the "Trebuchet MS" within

Notes

quotations marks. The final result sets the body color to light grey, and sets the font to ones that most users will have installed on there computer.

I have changed the way I layout my code, but you can arrange it in one line if you choose. I find that it is more readable if I spread each property to a separate line, with a 2 space indention.

Inheritance

When you nest one element inside another, the nested element will inherit the properties assigned to the containing element. Unless you modify the inner elements values independently.

Example: A font declared in the body will be inherited by all text in the file no matter the containing element, unless you declare another font for a specific nested element.

```
body {font-family: Verdana, serif;}
```

Now all text within the (X)HTML file will be set to Verdana.

If you wanted to style certain text with another font, like an h1 or a paragraph then you could do the following.

```
h1 {font-family: Georgia, sans-serif;}
p {font-family: Tahoma, serif;}
```

Now all <h1> tags within the file will be set to Georgia and all <p> tags are set to Tahoma, leaving text within other elements unchanged from the body declaration of Verdana.

There are instances where nested elements do not inherit the containing elements properties.

Example: If the body margin is set to 20 pixels, the other elements within the file will not inherit the body margin by default.

```
body {margin: 20px;}
```

Combining Selectors

You can combine elements within one selector in the following fashion.

```
h1, h2, h3, h4, h5, h6 {
  color: #009900;
  font-family: Georgia, sans-serif;
}
```

As you can see in the above code, I have grouped all the header elements into one selector. Each one is separated by a comma. The final result of the above code sets all headers to green and to the specified font. If the user does not have the first font I declared it will go to another sans-serif font the user has installed on their computer.

Comment tags

Comments can be used to explain why you added certain selectors within your css file. So as to help others who may see your file, or to help you remember what you we're thinking at a later date. You can add comments that will be ignored by browsers in the following manner.

```
/* This is a comment */
```

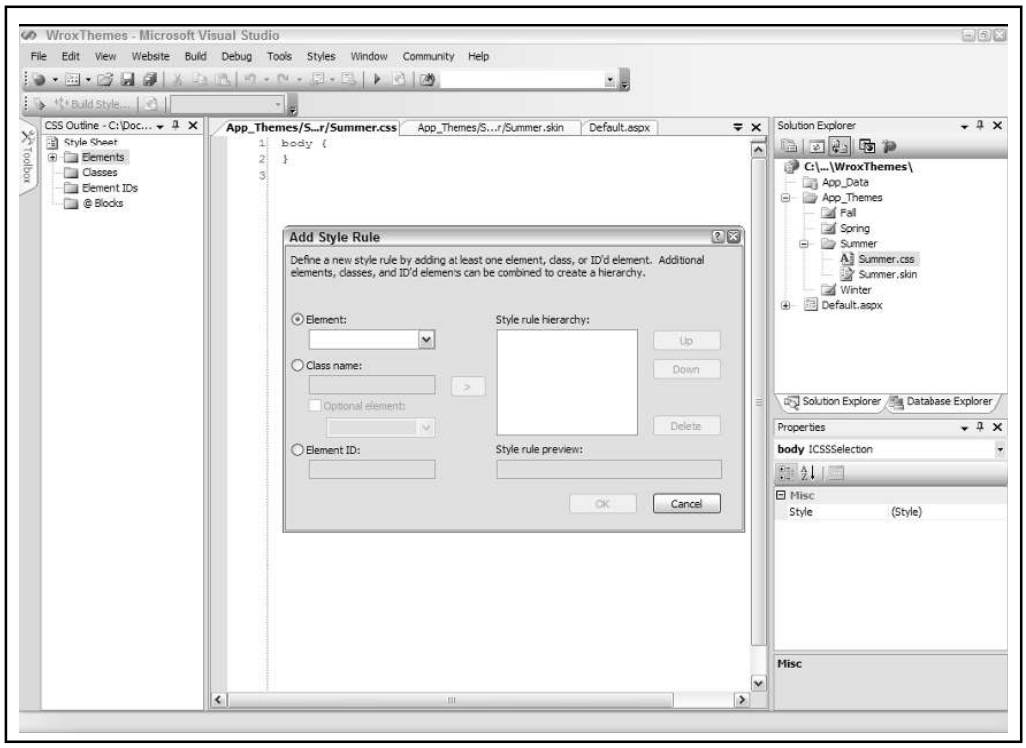
Notes

You will note that it begins with a / (forward slash) and than an * (asterisks) then the comment, then the closing tag which is just backward from the opening tag * (asterisks) then the / (forward slash).

In addition to the server control definitions that you create from within a .skin file, you can make further definitions using Cascading Style Sheets (CSS). You might have noticed, when using a .skin file, that you could define only the styles associated with server controls and nothing else. But developers usually use quite a bit more than server controls in their ASP.NET pages. For instance, ASP.NET pages are routinely made up of HTML server controls, raw HTML, or even raw text. At present, the Summer theme has only a Summer.skin file associated with it. Any other items have no style whatsoever applied to them.

For a theme that goes beyond the server controls, you must further define the theme style so that HTML server controls, HTML, and raw text are all changed according to the theme. You achieve this with a CSS file within your Themes folder.

It is rather easy to create CSS files for your themes when using Visual Studio 2005. Right-click the Summer theme folder and select Add New Item. In the list of options, select the option Style Sheet and name it Summer.css. The Summer.css file should be sitting right next to your Summer.skin file. This creates an empty .css file for your theme. I won't go into the details of how to make a CSS file using Visual Studio 2005 and the CSS creation tool because this was covered earlier in the book. The process is the same as in previous versions of Visual Studio. Just remember that the dialog that comes with Visual Studio 2005 enables you to completely define your CSS page with no need to actually code anything. A sample dialog is shown in figure.



To create a comprehensive theme with this dialog, you define each HTML element that might appear in the ASP.NET page. This can be a lot of work, but it's worth it in the end. For now, create a small CSS file that changes some of the non-server control items on your ASP.NET page. This CSS file is shown in Listing 10.5.

Notes

Listing 10.5: A CSS file with some definitions

```
body
{
font-size: x-small;
font-family: Verdana;
color: #004000;
}
A:link {
color: Blue;
text-decoration: none;
}
A:visited
{
color: Blue;
text-decoration: none;
}
A:hover {
COLOR: Red;
text-decoration: underline overline;
}
```

In this CSS file, four things are defined. First, you define text that is found within the <body> tag of the page (basically all the text). Plenty of text appears in a typical ASP.NET page that is not placed inside an <asp:Label> or <asp:Literal> tag. Therefore, you define how your text should appear; otherwise, your Web page may appear quite odd at times. In this case, a definition is in place for the size, the font family, and the color of the text. You make this definition the same as the one for the <asp:Label> server control in the Summer.skin file.

The next three definitions in this CSS file revolve around the <a> element (for hyperlinks). One cool feature that many Web pages use is responsive hyperlinks—or hyperlinks that change when you hover a mouse over them. The A:link definition defines what a typical link looks like on the page. The A:visited definition defines the look of the link if the end user has clicked on the link previously (without this definition, it is typically purple in IE). Then the A:hover definition defines the appearance of the hyperlink when the end user hovers the mouse over the link. You can see that not only are these three definitions changing the color of the hyperlink, but they are also changing how the underline is used. In fact, when the end user hovers the mouse over a hyperlink on a page using this CSS file, an underline and an overline appear on the link itself.

In CSS files, the order in which the style definitions appear in the .css file is important. This is an interpreted file—the first definition in the CSS file is applied first to the page, next the second definition is applied, and so forth. Some styles might change previous styles, so make sure your style definitions are in the proper order. For instance, if you put the A:hover style definition first, you would never see it. The A:link and A:visited definitions would supersede it because they are defined after it.

In working with your themes that include .css files, you must understand what they can and cannot do for you. For instance, examine an .aspx file that contains two text boxes—one text box created using a server control and another text box created using a typical <input> HTML element:

```
<asp:Textbox ID="TextBox1" Runat="server" /> 
<input type="text" />
```

Notes

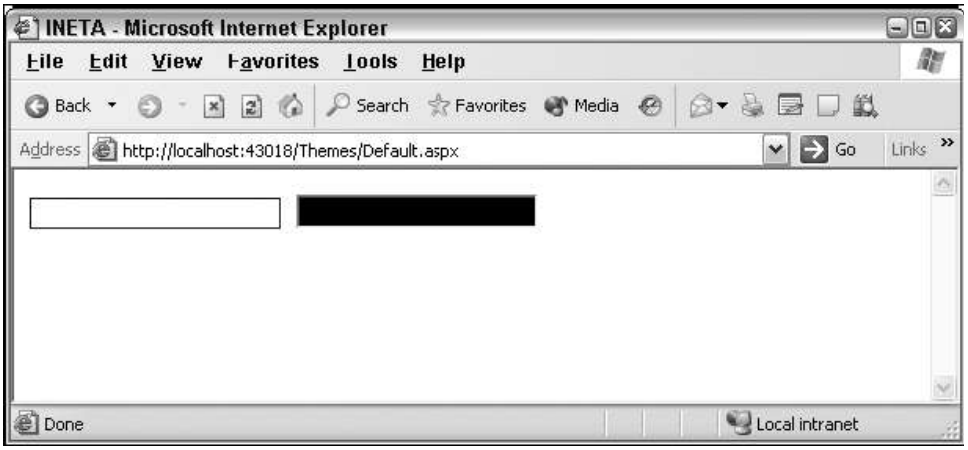
Suppose you have a definition for the TextBox server control in the .skin file:

```
<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"
BackColor="#ffffff" Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
BorderColor="#004000" Font-Bold="True" />
```

But, what if you also have a definition in your .css file for each <input> element in the ASP.NET page as shown here:

```
INPUT
{
background-color: black;
}
```

When you run the .aspx page with these kinds of style conflicts, the .skin file takes precedence over styles applied to every HTML element that is created using ASP.NET server controls regardless of what the .css file says. In fact, this sort of scenario gives you a page in which the <input> element that is created from the server control is white as defined in the .skin file and the second text box is black as defined in the .css file. This is shown in figure.

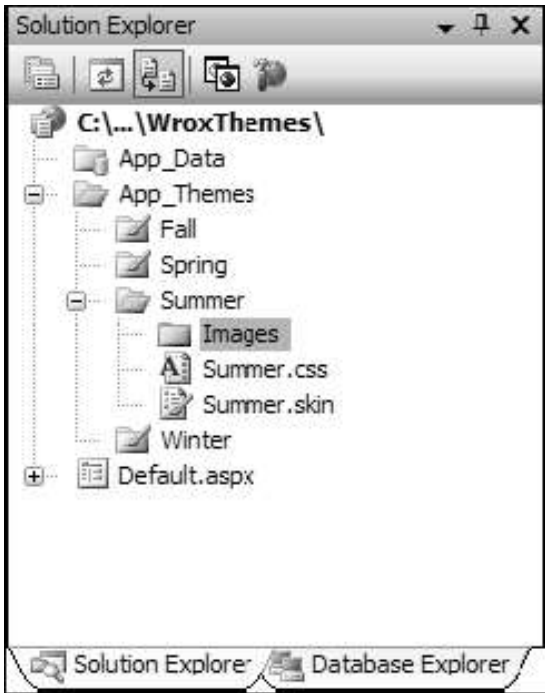


Having your themes include Images

Probably one of the coolest reasons why themes, rather than CSS, are the better approach for applying a consistent style to your Web page is that themes enable you to incorporate actual images into the style definitions.

A lot of controls use images to create a better visual appearance. The first step in incorporating images into your server controls that consistently use themes is to create an Images folder within the Themes folder itself, as illustrated in figure.

Notes



You have a couple of easy ways to use the images that you might place in this folder. The first is to incorporate the images directly from the .skin file itself. You can do this with the TreeView server control. The TreeView control can contain images used to open and close nodes for navigation purposes. You can place images in your theme for each and every TreeView control in your application. If you do so, you can then define the TreeView server control in the .skin file, as shown in Listing 10.6.

Listing 10.6: Using images from the theme folder in a TreeView server control

```
<asp:TreeView runat="server" BorderColor="#FFFFFF" BackColor="#FFFFFF"
ForeColor="#585880" Font-Size=".9em" Font-Names="Verdana"
LeafNodeImageURL="images\summer_iconlevel.gif"
RootNodeImageURL="images\summer_iconmain.gif"
ParentNodeImageURL="images\summer_iconmain.gif" NodeIndent="30"
CollapseImageURL="images\summer_minus.gif"
ExpandImageURL="images\summer_plus.gif">
...
</asp:TreeView>
```

When you run a page containing a TreeView server control, it is populated with the images held in the Images folder of the theme.

It's easy to incorporate images into the TreeView control. The control even specifically asks for an image location as an attribute. The new WebParts controls are used to build portals. Listing 10.7 is an example of a Web Part definition from a .skin file that incorporates images from the Images folder of the theme.

Listing 10.7: Using images from the theme folder in a WebPartZone server control

```
<asp:WebPartZone ID="WebPartZone1" runat="server"
DragHighlightColor="#6464FE" BorderStyle="double"
BorderColor="#E7E5DB" BorderWidth="2pt" BackColor="#F8F8FC"
```

Notes

```
cssclass="theme_fadeblue" Font-Size=".9em" Font-Names="Verdana">
<FooterStyle ForeColor="#585880" BackColor="#CCCCCC"></FooterStyle>
<HelpVerb ImageURL="images/SmokeAndGlass_help.gif"
checked="False" enabled="True" visible="True"></HelpVerb>
<CloseVerb ImageURL="images/SmokeAndGlass_close.gif"
checked="False" enabled="True" visible="True"></CloseVerb>
<RestoreVerb ImageURL="images/SmokeAndGlass_restore.gif"
checked="False" enabled="True" visible="True"></RestoreVerb>
<MinimizeVerb ImageURL="images/SmokeAndGlass_minimize.gif"
checked="False" enabled="True" visible="True"></MinimizeVerb>
<EditVerb ImageURL="images/SmokeAndGlass_edit.gif"
checked="False" enabled="True" visible="True"></EditVerb>
</asp:WebPartZone>
```

As you can see here, this series of toolbar buttons, which is contained in a WebPartZone control, now uses images that come from the aforementioned SmokeAndGlass theme. When this WebPartZone is then generated, the style is defined directly from the .skin file, but the images specified in the .skin file are retrieved from the Images folder in the theme itself.

Not all server controls enable you to work with images directly from the Themes folder by giving you an image attribute to work with. If you don't have this capability, you must work with the .skin file and the CSS file together. If you do, you can place your theme-based images in any element you want. Next is a good example of how to do this.

Place the image that you want to use in the Images folder just as you normally would. Then define the use of the images in the .css file. The continued SmokeAndGlass example in Listing 10.8 demonstrates this.

Listing 10.8: Part of the CSS file from SmokeAndGlass.css

```
theme_header {
background-image :url( images/smokeandglass_brownfadetop.gif);
}
.theme_highlighted {
background-image :url( images/smokeandglass_blueandwhitef.gif);
}
.theme_fadeblue {
background-image :url( images/smokeandglass_fadeblue.gif);
}
```

These are not styles for a specific HTML element; instead, they are CSS classes that you can put into any HTML element that you want. In this case, each CSS class mentioned here is defining a specific background image to use for the element.

After it is defined in the CSS file, you can utilize this CSS class in the .skin file when defining your server controls. Listing 10.9 shows you how.

Listing 10.9: Using the CSS class in one of the server controls defined in the .skin file

```
<asp:Calendar runat="server" BorderStyle="double" BorderColor="#E7E5DB"
BorderWidth="2" BackColor="#F8F7F4" Font-Size=".9em" Font-Names="Verdana">
<TodayDayStyle BackColor="#F8F7F4" BorderWidth="1" BorderColor="#585880"
ForeColor="#585880" />
<OtherMonthDayStyle BackColor="transparent" ForeColor="#CCCCCC" />
```

Notes

```
<SelectedDayStyle ForeColor="#6464FE" BackColor="transparent"
CssClass="theme_highlighted" />
<TitleStyle Font-Bold="True" BackColor="#CCCCCC" ForeColor="#585880"
BorderColor="#CCCCCC" BorderWidth="1pt" CssClass="theme_header" />
<NextPrevStyle Font-Bold="True" ForeColor="#585880"
BorderColor="transparent" BackColor="transparent" />
<DayStyle ForeColor="#000000"
BorderColor="transparent" BackColor="transparent" />
<SelectorStyle Font-Bold="True" ForeColor="#696969" BackColor="#F8F7F4" /
>
<WeekendDayStyle Font-Bold="False" ForeColor="#000000"
BackColor="transparent" />
<DayHeaderStyle Font-Bold="True" ForeColor="#585880"
BackColor="Transparent" />
</asp:Calendar>
```

This Calendar server control definition from a .skin file uses one of the earlier CSS classes in its definition. It actually uses an image that is specified in the CSS file in two different spots within the control (shown in bold). It is first specified in the <SelectedDayStyle> element. Here you see the attribute and value CssClass="theme_highlighted". The other spot is within the <TitleStyle> element. In this case, it is using theme_header. When the control is rendered, these CSS classes are referenced and finally point to the images that are defined in the CSS file.

It is interesting that the images used here for the header of the Calendar control don't really have much to them. For instance, the smokeandglass_brownfadetop.gif image that we are using for this example is simply a thin, gray sliver, as shown in Figure 10.10.

This very small image (in this case, very thin) is actually repeated as often as necessary to make it equal the length of the header in the Calendar control. The image is lighter at the top and darkens toward the bottom. Repeated horizontally, this gives a three-dimensional effect to the control. Try it out, and you get the result shown in figure.

Notes

10.6 Defining Multiple Skin Options

Using the themes technology in ASP.NET 2.0, you can have a single theme; but also, within the theme's .skin file, you can have specific controls that are defined in multiple ways. You can frequently take advantage of this feature within your themes. For instance, you might have text box elements scattered throughout your application, but you might not want each and every text box to have the same visual appearance. In this case, you can create multiple versions of the <asp:Textbox> server control within your .skin file. In Listing 10.10 you see how to create multiple versions of the <asp:Textbox> control in the .skin file from Listing 10.3.

Listing 10.10: The Summer.skin file, which contains multiple versions of the <asp:Textbox> server control

```
<asp:Label Runat="server" ForeColor="#004000" Font-Names="Verdana"
Font-Size="X-Small" />

<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"
Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
BorderColor="#004000" Font-Bold="True" />

<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Verdana"
Font-Size="X-Small" BorderStyle="Dotted" BorderWidth="5px"
BorderColor="#000000" Font-Bold="False" SkinID="TextboxDotted" />

<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Arial"
Font-Size="X-Large" BorderStyle="Dashed" BorderWidth="3px"
BorderColor="#000000" Font-Bold="False" SkinID="TextboxDashed" />

<asp:Button Runat="server" ForeColor="#004000" Font-Names="Verdana"
Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
BorderColor="#004000" Font-Bold="True" BackColor="#FFE0C0" />
```

In this .skin file, you can see three definitions in place for the TextBox server control. The first one is the same as before. Although the second and third definitions have a different style, they also contain a new attribute in the definition SkinID. To create multiple definitions of a single element, you use the SkinID attribute to differentiate among the definitions. The value used in the SkinID can be anything you want. In this case, it is TextboxDotted and TextboxDashed.

Note that no SkinID attribute is used for the first <asp:Textbox> definition. By not using one, you are saying that this is the default style definition to use for each <asp:Textbox> control on an ASP.NET page that uses this theme but has no pointer to a SkinID.

Take a look at a sample .aspx page that uses this .skin file in Listing 10.11.

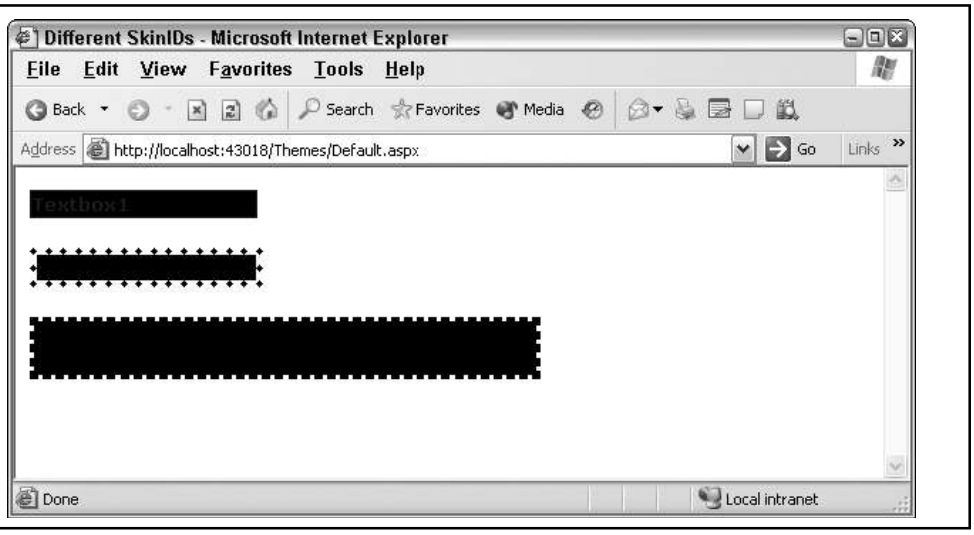
Listing 10.11: A simple .aspx page that uses the Summer.skin file with multiple textbox style definitions

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

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Different SkinIDs</title>
</head>
```

Notes

```
<body>
<form id="form1" runat="server">
<p>
<asp:Textbox ID="TextBox1" Runat="server">Textbox1</asp:Textbox>
</p><p>
<asp:Textbox ID="TextBox2" Runat="server"
SkinId="TextboxDotted">Textbox2</asp:Textbox>
</p><p>
<asp:Textbox ID="TextBox3" Runat="server"
SkinId="TextboxDashed">Textbox3</asp:Textbox>
</p>
</form>
</body>
</html>
```

This small .aspx page shows three text boxes, each of a different style. When you run this page, you get the results shown in figure.



The first text box doesn't point to any particular SkinID in the .skin file. Therefore, the default skin is used. As stated before, the default skin is the one in the .skin file that doesn't have a SkinID attribute in it. The second text box then contains skinid="TextboxDotted" and, therefore, inherits the style definition defined in the TextboxDotted skin in the Summer.skin file. The third text box takes the SkinID TextboxDashed and is also changed appropriately.

As you can see, it is quite simple to define multiple versions of a control that can be used throughout your entire application.

10.7 Programmatically Working with Themes

So far, you have seen examples of working with ASP.NET 2.0 themes in a declarative fashion, but you can also work with themes programmatically.

Assigning the Page's Theme Programmatically

To programmatically assign the theme to the page, use the construct shown in Listing 10.12.

Notes

Listing 10.12: Assigning the theme of the page programmatically

```
<script runat="server" language="vb">
Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
Page.Theme = Request.QueryString("ThemeChange")
End Sub
</script>
```

You must set the Theme of the Page property in or before the Page_PreInit event for any static controls that are on the page. If you are working with dynamic controls, set the Theme property before adding it to the Controls collection.

Assigning a Control's SkinID Programmatically

Another option is to assign a specific server control's SkinID property programmatically (Listing 10.13).

Listing 10.13: Assigning the server control's SkinID property programmatically

```
<script runat="server" language="vb">
Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
TextBox1.SkinID = "TextboxDashed"
End Sub
</script>
```

Again, you assign this property before or in the Page_PreInit event in your code.

Task Create an HTML layout which responds to the user clicking on a command button by displaying text within a text control.

10.8 Themes and Custom Controls

If you are building custom controls in an ASP.NET 2.0 world, understand that end users can also apply themes to the controls that they use in their pages. By default, your custom controls are theme-enabled whether your custom control inherits from Control or WebControl.

To disable theming for your control, you can simply use the Themeable attribute on your class. This is illustrated in Listing 10.14.

Listing 10.14:

```
Imports System.Web.UI
Namespace Wrox.ServerControls
<Themeable(False)> _
Public Class SimpleHello
Inherits System.Web.UI.Control
Private _name As String
Public Property Name() As String
Get
Return _name
End Get
Set(ByVal Value As String)
```

Notes

```
_name = Value
End Set
End Property

Protected Overrides Sub Render(ByVal controlOutput As _
HtmlTextWriter)
controlOutput.Write("Hello " + Name)

End Sub



End Class
End Namespace
```

You can also disable theming for the individual properties that might be in your custom controls. You do this as illustrated in Listing 10.15.

Listing 10.15: Disabling theming for properties in your custom controls

```
Imports System.Web.UI

Namespace Wrox.ServerControls

Public Class SimpleHello
Inherits System.Web.UI.Control

Private _myValue As String

<Themeable(False)> _
Public Property MyCustomProperty() As String
Get
Return _myValue
End Get
Set(ByVal Value As String)
_myValue = Value
End Set
End Property

End Class
End Namespace
```

Notes

Case Study

Retailers turn to Smart Carts

The retail industry is going through an extraordinary metamorphosis as transactions are increasingly supported by a wide variety of digital technologies. The previous units provided many examples of businesses expanding to the Web to reach more customers. New forms of transaction data collection are also evident in brick-and-mortar stores. For example, consider the rapidly expanding number of self-service checkout systems in popular grocery stores, department stores, super discount stores, home warehouse stores, and even fast food restaurants.

Fujitsu calls it the Pervasive Retailing Environment: the use of digital technologies to integrate wired and wireless network devices to facilitate transactions in retail stores.

Self-serve check-outs are only the tip of the iceberg. Soon customers will have access to product information from any location in the store through devices like Fujitsu's U-Scan Shopper. Mounted on a shopping cart, the U-Scan Shopper is a rugged wireless computer with an integral bar code scanner. The device provides services to shoppers as well as retailer.

The device reduces checkout time by allowing customers to scan and bag items themselves as they pick them off the selves. Shoppers can view the running total to see exactly how much is being spent as they shop. No more surprises at the checkout counter. If an item is missing a price, the device can be used as a price-checker. Consumers can also use the

U-Scan Shopper to place orders with departments in the store for pickup. For example, you can place a deli or prescription order when you arrive at the store and pick it up at the deli counter or pharmacy. The U-Scan Shopper also provides a store directory so you can easily find the department or goods you want.

U-Scan devices are integrated into the store network and internet. This means customers can upload a shopping list to the store's Web site before leaving home, and then download the list to the shopping cart upon arriving at the store. When shopping is completed, the U-Scan device uploads information to the self-serve checkout and the shopper is out the door after a quick swipe of a debit or credit card.

For retailers, the U-Scan device offers what Fujitsu calls "true 1:1 marketing" the enables personalized in-store advertisements campaigns that are relevant both to shoppers' preferences and to their location in the store. Location is determined by shelf-mounted, battery-powered infrared transmitters that track the movement of U-Scan devices through the store. As a shopper passes the condiments aisle, for example, the shopping cart display might post a message stating. "It has been over a month since you purchased mustard. If you want to pick some up today, turn down this aisle." A retailer can offer special deals to each consumer. For example, as a shopper passes the condiments aisle, message on the U-Scan device might state, "You have just won an electronic coupon for $0.89 off mustard. Turn now to take advantage of this special deal!" The 89 cents would be deducted as the item is scanned on the U-Scan device.

Questions

1. What transaction processing services does the U-Scan Shopper provide for consumers?
2. How does U-Scan technology provide retailers with a competitive advantage? Why might you choose a U-Scan store over one without U-Scan devices?

Notes

10.9 Summary

- With the addition of themes and skins in ASP.NET 2.0, it has become quite easy to apply a consistent look and feel across your entire application.
- Remember that themes can contain only simple server control definitions in a .skin file or elaborate style definitions, which include not only .skin files, but also CSS style definitions and even images.
- You can use themes in conjunction with the new personalization features that ASP.NET 2.0 provides.
- This can enable your end users to customize their experiences by selecting their own themes.
- Your application can present a theme just for them, and it can remember their choices through the APIs that are offered in ASP.NET 2.0.

10.10 Keywords

Image Well: A collection of images of the same size and color depth that are stored as a row of images in a single bitmap.

Internal Style Sheet: A style sheet contained in an ASP.NET mobile Web Forms page.

Skin File: A file containing one or more control properties that define how the controls should look.

Stylesheets: Data files used to express how the structured content of a document should be presented on a particular physical medium (e.g., printed pages, Web browser, hand-held device, etc.). Details of the presentation include font style, lay out, and pagination.

Theme: In ASP.NET, a collection of control properties, stylesheets, and images that can be applied as a unit to a page or Web site to define an overall appearance.

10.11 Self Assessment

Fill in the blanks:

1. Themes are similar to ..... in that they enable you to define visual styles for your Web pages.
2. The black background color and the color of the text in the text box are specified directly in the control itself with the use of the BackColor and ..... attributes.
3. The ..... attribute works the same as the Theme attribute in that it can be used to apply a theme to a page.
4. A ..... is a definition of styles applied to the server controls in your ASP.NET page.
5. If you specify the theme in the ..... file, you don't need to define the theme again in the Page directive of your ASP.NET pages.
6. The skin file can have any name, but it must have a .skin file extension.

State whether the following statements are True or Fa lse

7. BackColor and ForeColor settings remain in place, even if they are explicitly defined in the theme.

Notes

8. Themes are not a great way to easily apply a consistent look-and-feel across your entire application.
9. Skins can work in conjunction with CSS files or images.
10. ASP.NET pages are not routinely made up of HTML server controls.

10.12 Review Questions

1. What do you need to do to enable user profiles in your site?
2. What modifications do you need to make to web.config to specify what profile information you want to retain for your users?
3. What property of the Profile object do you use to determine whether a user is logged in?
4. How do you retain profile information that's not saved as a string?
5. How do you allow for profile information for a user without logging in?
6. How do you indicate that a specific piece of profile information should be retained for an anonymous user?
7. When a user logs in, how do you transfer any personalization data that the user might have entered as an anonymous user?
8. What's the difference between style sheet themes and customization themes?
9. Where do you specify settings for a skin?
10. How do you specify what theme to use on a page?

Answers: Self Assessment

1.	Cascading Style Sheets (CSS)	2.	ForeColor
3.	StylesheetTheme	4.	skin
5.	Web.config	6.	True
7.	True	8.	False
9.	True	10.	False

10.13 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Notes

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Notes

Unit 11: Master Page Basics

CONTENTS

Objectives
Introduction
11.1 Need of Master Pages
11.2 The Basics of Master Pages
11.3 Coding a Master Page
11.4 Coding a Content Page
11.5 Mixing Page Types and Languages
11.6 Specifying which Master Page to Use
11.7 Specifying Default Content in the Master Page
11.8 Programmatically Assigning the Master Page
11.9 Caching with Master Pages
11.10 Summary
11.11 Keywords
11.12 Self Assessment
11.13 Review Questions
11.14 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe need of master pages
- Know basics of master pages
- Explain coding of master page

Introduction

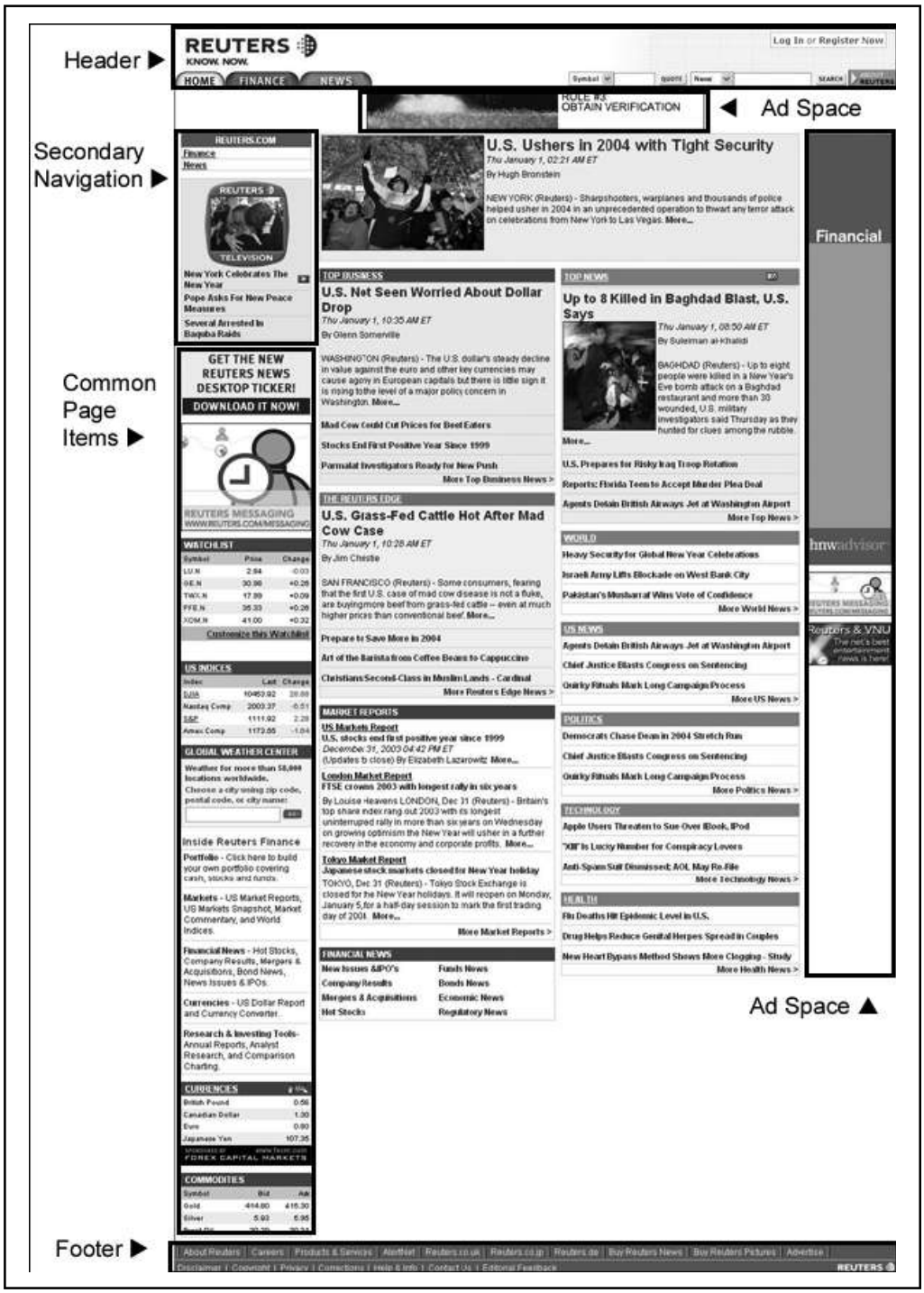
Visual inheritance is a great new enhancement to your Web pages provided by new additions to ASP.NET. In effect, you can create a single template page that can be used as a foundation for any number of ASP.NET content pages in your application. These templates, called master pages, increase your productivity by making your applications easier to build and easier to manage after they are built. Visual Studio 2005 includes full designer support for master pages, making the developer experience richer than ever before. This unit takes a close look at how to utilize master pages to the fullest extent in your applications and begins by explaining the advantages of master pages.

11.1 Need of Master Pages

Most Web sites today have common elements used throughout the entire application or on a majority of the pages within the application. For instance, if you look at the main page of the

Notes

Reuters News Web site (found at www.reuters.com), you see common elements that are used throughout the entire Web site. These common areas are labeled in figure.



In above figure, notice a header section, a navigation section, and a footer section on the page. In fact, nearly every page within the entire application uses these same elements. Even before master pages, you had ways to put these elements into every page; but in most cases, doing so posed difficulties.

Notes

Some developers simply copy and paste the code for these common sections to each and every page that requires them. This works, but it's rather labor intensive. But, if you use the copy-and paste method, whenever a change is required to one of these common sections of the application, you have to go into each and every page and duplicate the change. That's not much fun and an ineffective use of your time!

In the days of Classic Active Server Pages, one popular option was to put all the common sections into what was called an include file. You could then place this file within your page like this:

```
<!-- #include virtual="/myIncludes/header.asp" -->
```

The problem with using include files was that you had to take into account the newly opened HTML tags in the header include file. These tags had to be closed in the main document or in the footer include file. It was usually difficult to keep all the HTML tags in order, especially if multiple people worked on a project. Web pages sometimes displayed strange results because of inappropriate or nonexistent tag closings or openings. It was also difficult to work with include files in a visual designer. Using include files didn't allow the developer to see the entire page as it would appear in a browser. The developer ended up developing the page in sections and hoping that the pieces would come together as planned. Many hours were wasted "chasing tables" opened in an include file and possibly closed later!

With the introduction of ASP.NET 1.0 in 2000, developers started using user controls to encapsulate common sections of their Web pages. For instance, you could build a Web page that included header, navigation, and footer sections by simply dragging and dropping these sections of code onto each page that required them.

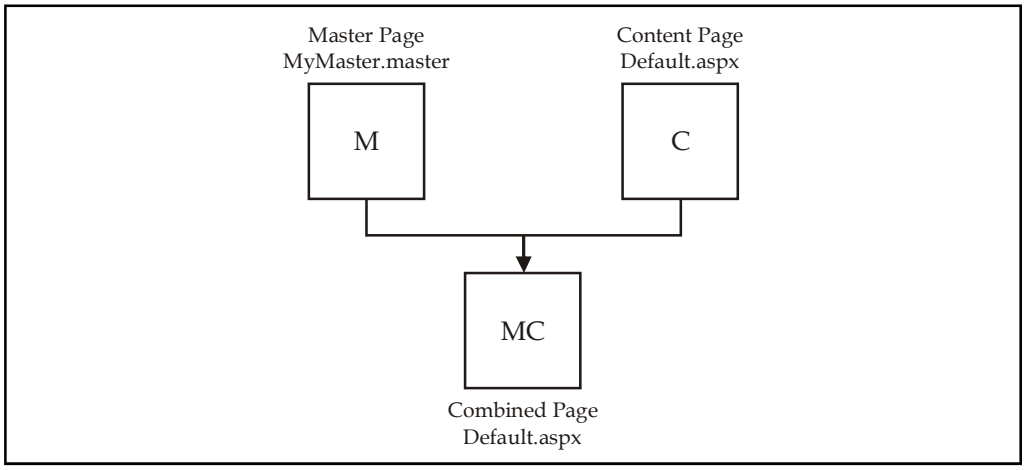
This technique worked, but it also raised some issues. Before Visual Studio 2005, user controls caused problems similar to those related to include files. When you worked in the design view of your Web page, the common areas of the page displayed only as gray boxes in Visual Studio .NET 2002 and 2003. This made it harder to build a page. You could not visualize what the page you were building actually looked like until you compiled and ran the completed page in a browser. User controls also suffered from the same problem as include files—you had to match up the opening and closing of your HTML tags in two separate files. Personally, we prefer user controls over include files, but user controls aren't perfect template pieces for use throughout an application. Visual Studio 2005 corrects some of the problems by rendering user-control content in the design view. User controls are ideal if you are including only small sections on a Web page; they are still rather cumbersome, however, when working with larger page templates.

In light of the issues with include files and user controls, the ASP.NET team developed the idea of master pages an outstanding new way of applying templates to your applications. They inverted the way the developer attacks the problem. Master pages live outside the pages you develop, while user controls live within your pages and are doomed to duplication. These master pages draw a more distinct line between the common areas that you carry over from page to page and the content areas that are unique on each page. Working with master pages is easy and fun. Look at some of the basics of master pages in ASP.NET

11.2 The Basics of Master Pages

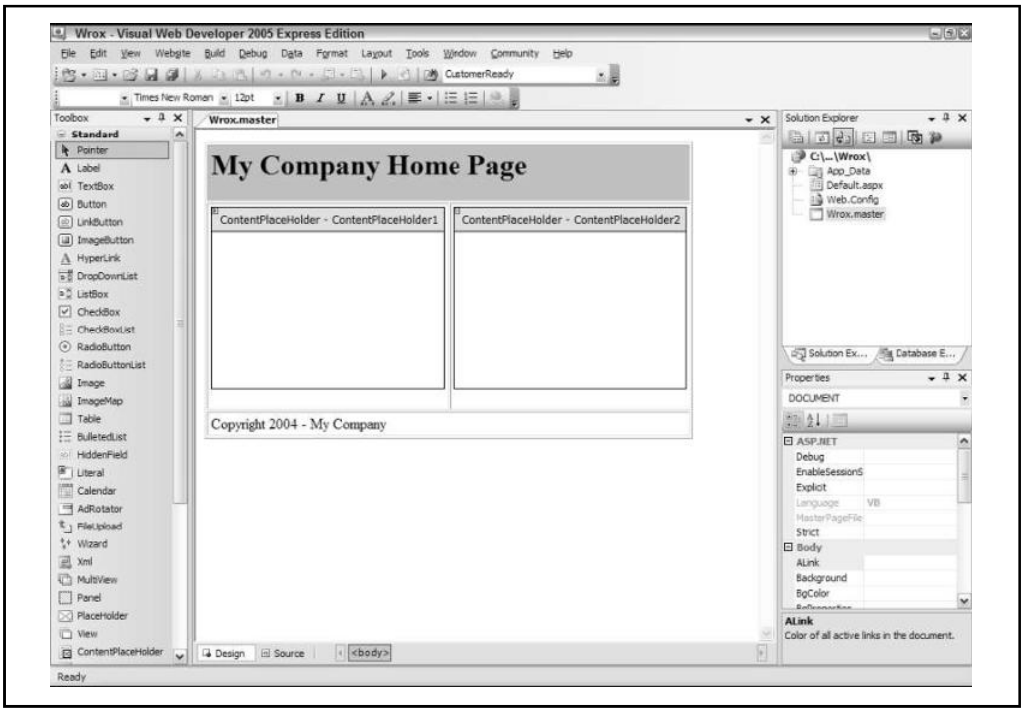
Master pages are an easy way to provide a template that can be used by any number of ASP.NET pages in your application. In working with master pages, you create a master file that is the template referenced by a subpage or content page. Master pages use a .master file extension, whereas content pages use the .aspx file extension you're used to; but content pages are declared as such within the file's Page directive.

Notes



Put anything you want to share within the template in the .master file. This can include the header, navigation, and footer sections used across the Web application. The content page then contains all the page content except for the master page's elements. At runtime, the ASP.NET engine combines these elements into a single page for the end user. Figure shows a diagram of how this process works.

One of the nice things about working with master pages is that you can visually see the template in the IDE when you are creating the content pages. Because you can see the entire page while you are working on it, it is much easier to develop content pages that use a template. While you are working on the content page, all the templated items are shaded gray and are not editable. The only items that are alterable are clearly shown in the template. These workable areas, called content areas, originally are defined in the master page itself. Within the master page, you specify the areas of the page that the content pages can use. You can have more than one content area in your master page if you want. Figure shows the master page with a couple of content areas shown.



Notes

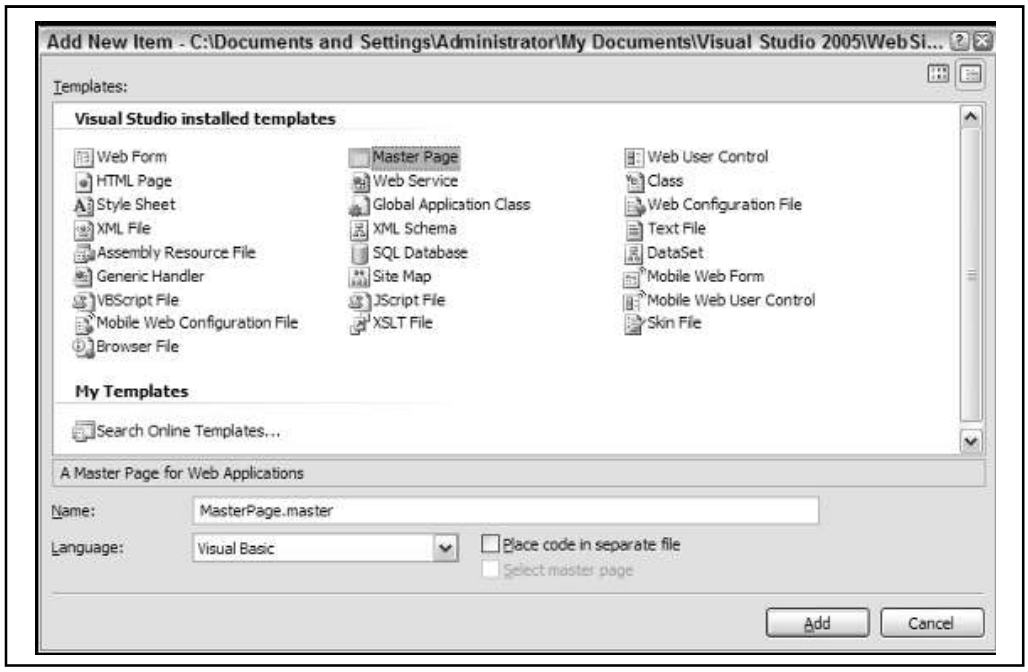
With the release of ASP.NET 2.0, master pages are possible because the .NET Framework 2.0 now supports partial classes. This is the capability to take two classes and merge them into a single class at runtime. Using this new capability, the ASP.NET engine takes two page classes and brings them together into a single page class at runtime.

Companies and organizations will find using master pages ideal, as the technology closely models their typical business requirements. Many companies have a common look and feel that they apply across their intranet. They can now provide the divisions of their company with a .master file to use when creating a department's section of the intranet. This process makes it quite easy for the company to keep a consistent look and feel across its entire intranet.

11.3 Coding a Master Page

Now look at building the master page shown in figure. You can create one in any text-based editor, such as Notepad or Visual Web Developer Express Edition, or you can use the new Visual Studio 2005. In this unit, you see how to use Visual Web Developer.

Master pages are added to your projects in the same way as regular .aspx pages choose the Master Page option when you add a new file to your application, as shown in figure.



Because it's just like any other .aspx page, the Add New Item dialog enables you to choose from a master page using the inline coding model or a master page that places its code in a separate file. Not placing your server code in a separate file means that you use the inline code model for the page you are creating. This option creates a single .master page. Choosing the option to place your code in a separate file means that you use the new code-behind model with the page you are creating. Selecting the check box Place Code In Separate File creates a single .master page, along with an associated .master .vb or .master.cs file.

A sample master page that uses the inline-coding model is shown in Listing 11.1.

Notes

Listing 11.1: A sample master page

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

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
	<title>My Company Master Page</title>
</head>
<body>
	<form id="form1" runat="server">
		<table cellpadding="3" border="1">
			<tr bgcolor="silver">
				<td colspan="2">
					<h1>My Company Home Page</h1>
				</td>
			</tr>
			<tr>
				<td>
				< a s p : C o n t e n t P l a c e H o l d e r
				ID="ContentPlaceHolder1"
					Runat="server">
					</asp:ContentPlaceHolder>
				</td>
				<td>
				< a s p : C o n t e n t P l a c e H o l d e r
				ID="ContentPlaceHolder2"
					Runat="server">
					</asp:ContentPlaceHolder>
				</td>
			</tr>
			<tr>
				<td colspan="2">
				Copyright 2006 - My Company
				</td>
			</tr>
		</table>
	</form>
</body>
</html>
```

This is a simple master page. The great thing about creating master pages in Visual Studio 2005 is that you can work with the master page in code view, but you can also switch over to design view to create your master pages.

Notes

Start by reviewing the code for the master page. The first line is the directive:

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

Instead of using the Page directive, as you would with a typical .aspx page, you use the Master directive for a master page. This master page uses only a single attribute, Language. The Language attribute's value here is VB, but of course, you could also use C# if you are building a C# master page.

You code the rest of the master page just as you would any other .aspx page. You can use server controls, raw HTML and text, images, events, or anything else you normally would use for any .aspx page. This means that your master page can have a Page_Load event as well or any other event that you deem appropriate.

In the code shown in Listing 11.1, notice the use of a new server control the <asp:ContentPlaceHolder> control. This control defines the areas of the template where the content page can place its content:

```
<tr>
<td>
<asp:ContentPlaceHolder ID="ContentPlaceHolder1"
Runat="server">
</asp:ContentPlaceHolder>
</td>
<td>
<asp:ContentPlaceHolder ID="ContentPlaceHolder2"
Runat="server">
</asp:ContentPlaceHolder>
</td>
</tr>
```

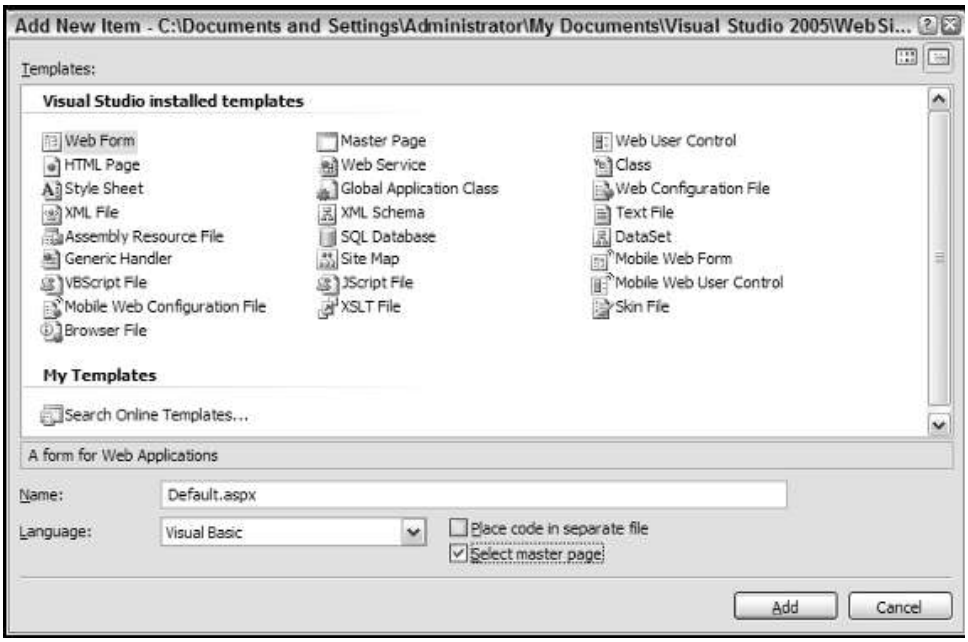
In the case of this master page, two defined areas exist where the content page can place content. Our master page contains a header and a footer area. It also defines two areas in the page where any inheriting content page can place its own content. Look at how a content page uses this master page.

11.4 Coding a Content Page

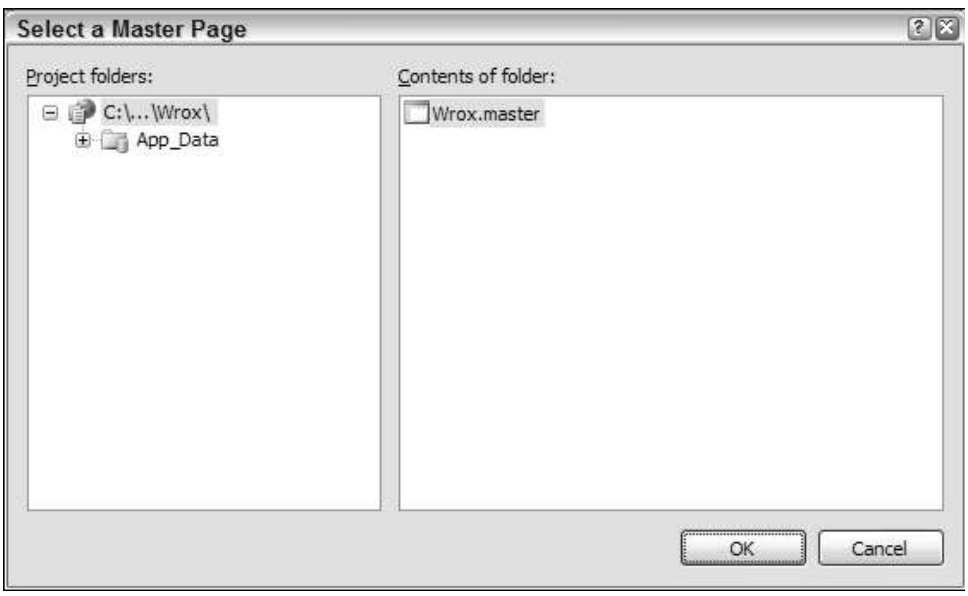
Now that you have a master page in place in your application, you can use this new template for any content pages in your application. Right-click the application in the Solution Explorer and choose Add New Item to create a new content page within your application.

To create a content page or a page that uses this master page as its template, you select a typical Web Form from the list of options in the Add New Item dialog. Instead of creating a typical Web Form, however, you check the Select Master Page check box. This gives you the option of associating this Web Form later to some master page. The Add New Item dialog is shown in figure.

Notes



After you name your content page and click the Add button in the Add New Item dialog, you are presented with the Select A Master Page dialog, as shown in figure.



This dialog allows you to choose the master page from which you want to build your content page. You choose from the available master pages that are contained within your application. For this example, select the new master page that you created in Listing 11.1 and click the OK button. This creates the content page. The created page is a simple .aspx page with only a single line of code contained in the file, as shown in Listing 11.2.

Notes

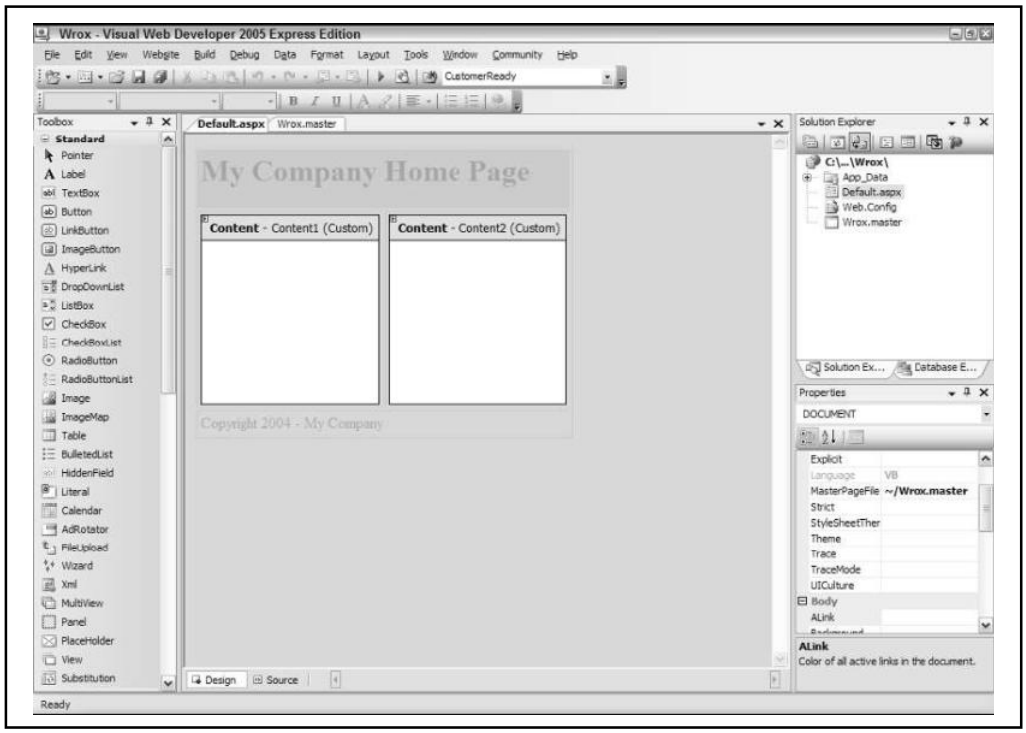
Listing 11.2: The created content page

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" Title="Untitled Page"
%>
```

This content page is not much different from the typical .aspx page you coded in the past. The big difference is the inclusion of the MasterPageFile attribute within the Page directive. The use of this attribute indicates that this particular .aspx page inherits from another page. The location of the master page within the application is specified as the value of the MasterPageFile attribute.

The other big difference is that it contains neither the <form id="form1" runat="server"> tag nor any opening or closing HTML tags that would normally be included in a typical .aspx page.

This content page may seem simple, but if you switch to the design view within Visual Studio 2005, you see the power of using content pages. What you get with visual inheritance is shown in figure.



In above figure, you can see that just by using the MasterFilePage attribute in the Page directive, you are able to visually inherit everything that the Wrox.master file exposes. All the common areas defined in the master page are shown in gray, whereas the content areas that you specified in the master page using the <asp:ContentPlaceHolder> server control are shown clearly and available for additional content in the content page. You can add any content to these defined content areas as if you were working with a regular .aspx page. An example of using this .master page for a content page is shown in Listing 11.3.

Listing 11.3: The content page that uses Wrox.master

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>

<script runat="server" language="vb">
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
```

Notes

```
Label1.Text = "<b>Hello " & TextBox1.Text & "!</b>"
End Sub
</script>
<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
Runat="server">
<b>Enter your name:</b><br />
<asp:Textbox ID="TextBox1" Runat="server" />
<br />
<br />
<asp:Button ID="Button1" Runat="server" Text="Submit"
OnClick="Button1_Click" /><br />
<br />
<asp:Label ID="Label1" Runat="server" />
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder2"
Runat="server">
<asp:Image ID="Image1" Runat="server" ImageUrl="wrox.gif" />
</asp:content>
```

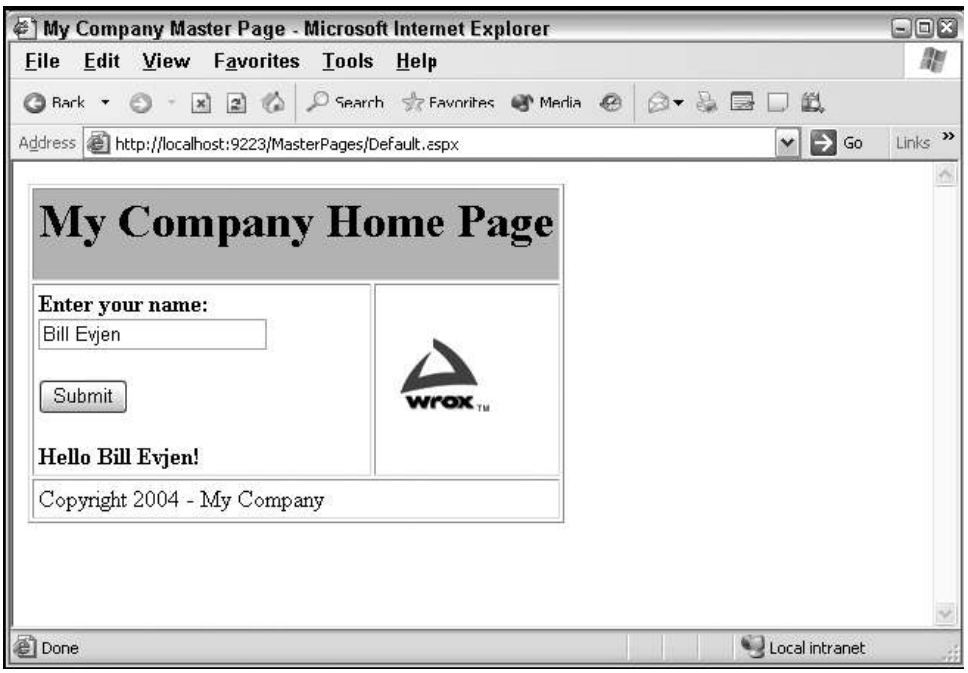
Right away you see some differences. As stated before, this page has no <form id="form1" runat="server"> tag nor any opening or closing HTML tags. These tags are not included because they are located in the master page. Also notice a new server control—the <asp:Content> server control.

```
<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
Runat="server">
...
</asp:Content>
```

The <asp:Content> server control is a defined content area that maps to a specific <asp:ContentPlaceHolder> server control on the master page. In this example, you can see that the <asp:Content> server control maps itself to the <asp:ContentPlaceHolder> server control in the master page that has the ID of ContentPlaceHolder1. Within the content page, you don't have to worry about specifying the location of the content because this is already defined within the master page. Therefore, your only concern is to place the appropriate content within the provided content sections, allowing the master page to do most of the work for you.

Just as when you work with any typical .aspx page, you can create any event handlers for your content page. In this case, you are using just a single event handler the button-click when the end user submits the form. The created .aspx page that includes the master page and content page material is shown in figure.

Notes



Task

Create an external style sheet for two HTML documents. Change the style sheet styles and reload the web pages to see that the changes are implemented in both.

11.5 Mixing Page Types and Languages

One interesting point: When you use master pages, you are not tying yourself to a specific coding model (inline or code-behind), nor are you tying yourself to the use of a specific language. You can feel free to mix these elements within your application knowing that they all work well.

You could use the master page created earlier, knowing that it was created using the inline-coding model, and then build your content pages using the code-behind model. Listing 11.4 shows a content page created using a Web Form that uses the code-behind option.

Listing 11.4: A content page that uses that code-behind model

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" AutoEventWireup="false"
CodeFile="MyContentPage.aspx.vb" Inherits="MyContentPage" %>
<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
Runat="server">
<b>Enter your name:</b><br />
<asp:Textbox ID="TextBox1" Runat="server" />
<br />
<br />
<asp:Button ID="Button1" Runat="server" Text="Submit"
```

Notes

```
OnClick="Button1_Click" /><br />
<br />
<asp:Label ID="Label1" Runat="server" />
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder2"
Runat="server">
<asp:Image ID="Image1" Runat="server" ImageUrl="ineta.JPG" />
</asp:Content>
```

Even though the master page is using the inline-coding model, you can easily create content pages (such as the page shown in Listing 11.4) that use the code-behind model. The pages will still work perfectly.

Not only can you mix the coding models when using master pages, you can also mix the programming languages you use for the master or content pages. Just because you build a master page in C# doesn't mean that you are required to use C# for all the content pages that use this master page. You can also build content pages in Visual Basic. For a good example, create a master page in C# that uses the Page_Load event handler and then create a content page in Visual Basic. Once it's complete, run the page. It works perfectly well. This means that even though you might have a master page in one of the available .NET languages, the programming teams that build applications from the master page can use whatever .NET language they want. You have to love the openness that the .NET Framework offers!

11.6 Specifying which Master Page to Use

You just observed that it is pretty easy to specify at page level which master page to use. In the Page directive of the content page, you simply use the MasterPageFile attribute:

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>
```

Besides specifying the master page that you want to use at the page level, you have a second way to specify which master page you want to use in the Web.config file of the application. This is shown in Listing 11.5.

Listing 11.5: Specifying the master page in the Web.config file

```
<configuration>
<system.web>
<pages masterPageFile="~/Wrox.master" />
</system.web>
</configuration>
```

Specifying the master page in the Web.config file causes every single content page you create in the application to inherit from the specified master page. If you declare your master page in the Web.config file, you can create any number of content pages that use this master page. Once specified in this manner, the content page's Page directive can then be constructed in the following manner:

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

Notes

You can easily override the application-wide master page specification by simply declaring a different master page within your content page:

```
<%@ Page Language="VB" MasterPageFile="~/MyOtherCompany.master" %>
```

By specifying the master page in the Web.config, you are really not saying that you want all the .aspx pages to use this master page. If you create a normal Web Form and run it, ASP.NET will know that the page is not a content page and will run the page as a normal .aspx page.

If you want to apply the master page template to only a specific subset of pages (such as pages contained within a specific folder of your application), you can use the <location> element within the Web.config file, as illustrated in Listing 11.6.

Listing 11.6: Specifying the master page for a specific folder in the Web.config file

```
<configuration>

<location path="AdministrationArea">
<system.web>
<pages masterPageFile="~/WroxAdmin.master" />
</system.web>
</location>

</configuration>
```

With the addition of this <location> section in the Web.config file, you have now specified that a specific older (Administration Area) will use a different master file template. This is done using the path attribute of the <location> element. The value of the path attribute can be a folder name as shown, or it can even be a specific page such as AdminPage.aspx.

11.7 Specifying Default Content in the Master Page

As you have seen, the master page enables you to specify content areas that the content page can use. Master pages can consist of just one content area, or they can be made up of multiple content areas. The nice thing about content areas is that when you create a master page, you can specify default content for the content area. This default content can then be left in place and utilized by the content page if you choose not to override it. Listing 11.7 shows a master page that specifies some default content within a content area.

Listing 11.7: Specifying default content in the master page

```
<%@ Master Language="VB" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>My Company</title>
</head>
<body>
<form id="form1" runat="server">
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" Runat="server">
Here is some default content
</asp:ContentPlaceHolder><p>
```

Notes

```
<asp:ContentPlaceHolder ID="ContentPlaceHolder2" Runat="server">
Here is some more default content
</asp:ContentPlaceHolder></p>
</form>
</body>
</html>
```

To place default content within one of the content areas of the master page, you simply put it in the ContentPlaceHolder server control on the master page itself. Any content page that inherits this master page also inherits the default content. Listing 11.8 shows a content page that overrides just one of the content areas from this master page.

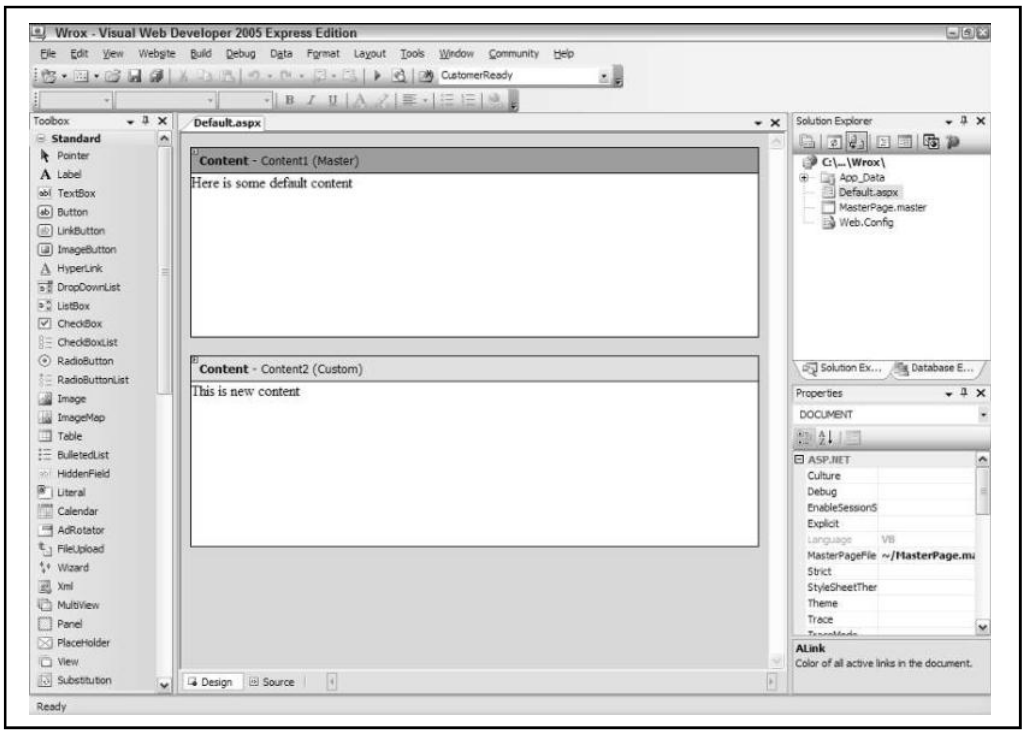
Listing 11.8: Overriding some default content in the content page

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" %>

<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder2"
Runat="server">
This is new content
</asp:Content>
```

This code creates a page with one content area that shows content coming from the master page itself, in addition to other content that comes from the content page.

The other interesting point when you work with content areas in the design mode of Visual Studio 2005 is that the smart tag allows you to work easily with the default content.

Notes



When you first start working with the content page, the option in the smart tag is to create new content. This option enables you to override the master page content and insert your own defined content. After you have placed some custom content inside the content area, the smart tag shows a different option Default to Master's Content. This option enables you to return the default content that the master page exposes to the content area and to erase whatever content you have already placed in the content area thereby simply returning to the default content.

11.8 Programmatically Assigning the Master Page

From any content page, you can easily assign a master page programmatically. You assign the master page to the content page through the use of the Page.MasterPageFile property. This can be used regardless of whether another master page is already assigned in the @Page directive.

To accomplish this, you use this property through the Page_PreInit event. The Page_PreInit event is the earliest point in which you can access the Page lifecycle. For this reason, this is where you need to assign any master page that is used by any content pages. The Page_PreInit is an important event to make note of when you are working with master pages, as this is the only point where you can affect both the master and content page before they are combined into a single instance. Listing 11.9 illustrates how to assign the master page programmatically from the content page.

Listing 11.9: Using Page_PreInit to assign the master page programmatically

```
<%@ Page Language="VB" %>
<script runat="server">
Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
Page.MasterPageFile = "~/MyMasterPage.master"
```

Notes

```
End Sub
</script>
```

In this case, when the page is dynamically being generated, the master page is assigned to the content page in the beginning of the page construction process. It is important to note that the content page must have the expected Content controls; otherwise an error is thrown.

Task

Create an onscreen clock accurate to the second. The time display format is not important.

11.9 Caching with Master Pages

When working with typical .aspx pages, you can apply output caching to the page by using the following construct (or variation thereof):

```
<%@ OutputCache Duration="10" Varybyparam="None" %>
```

This caches the page in the server's memory for 10 seconds. Many developers use output caching to increase the performance of their ASP.NET pages. It also makes a lot of sense for use on pages with data that doesn't become stale too quickly.

How do you go about applying output caching to ASP.NET pages when working with master pages? First, you cannot apply caching to just the master page. You cannot put the OutputCache directive on the master page itself. If you do so, on the page's second retrieval, you get an error because the application cannot find the cached page.

To work with output caching when using a master page, stick the OutputCache directive in the content page. This caches both the contents of the content page as well as the contents of the master page (remember, it is just a single page at this point). The OutputCache directive placed in the master page does not cause the master page to produce an error, but it won't be cached. This directive works in the content page only.

Case Study

Automotive Data Pioneer Provides Timely, Interactive Web and Mobile Content Delivery

Kelley Blue Book is one of the most familiar and enduring names in the automotive industry. First published in 1926—just as Ford was phasing out the legendary Model T and replacing it with the Model A—the "little blue book" became the standard, authoritative reference on vehicle pricing for car makers, dealers, financial institutions, and individual buyers and sellers.

In the ensuing decades, the publication which was first offered in leaflet form and later as an actual book with a royal-blue cover could be found across the United States in dog-eared editions used by those involved in buying or selling cars. In the mid-1990s, Kelley Blue Book began offering its publication online at www.kbb.com . While the physical book is still published, the online site has increasingly become the source that buyers turn to first for information about vehicles.

Contd...

Notes

"Kelley Blue Book's legacy has always been the little blue book," says Justin Yaros, Executive Vice President of Product Design and Development for Kelley Blue Book. "We still do sell the book, and it is used in the market. But more and more, we find that our customers, whether consumers, dealers, or industry partners, are relying upon the technical solutions and the electronic information that we provide."

The growth of the Internet has been a boon to Kelley Blue Book, due to the broad reach of the Web and the ability to offer current information faster. However, the fast pace of the Internet has also introduced challenges for the company. Not only do customers expect information that is always up-to-date, they also expect the site to evolve with frequent, innovative new features.

"The market has become increasingly more competitive for us because there are other players. Traditionally, there had only been a few; now there are many," Yaros says. "So, we have to do what we do better. We have to do it faster, we have to do it more accurately, and we have to do it with greater imagination."

For Kelley Blue Book, just having a Web site is not enough. The advent of different content delivery channels such as mobile Web content and and texting in online social networking, as well as the need to rapidly acquire business intelligence in response to shifting consumer and market needs, led the company to rethink its IT and business models.

"For example, we realized that we had to figure out the social networking space to improve our marketing effectiveness and the overall customer experience," says Andy Lapin, Director of Enterprise Architecture at Kelley Blue Book. "Many people using the Internet don't send a link to an article in an e-mail. Rather, they'll do things like share a link on Twitter or Facebook. We needed to be able to able to serve that market as well as the more traditional online market of people who come directly to our Web site, get the information they need, and leave."

Even serving the large base of typical Web visitors has become more complex. A consumer who reviews car options from a desktop or laptop computer during the week oftentimes would like to access the same information on a mobile phone when visiting dealers in person on the weekend.

"Our customers, whether consumers or dealers, need to know that they can access our information through multiple plat-forms," says Yaros. "Their desktop and mobile devices have to work together seamlessly, and both need to provide a rich media experience. Some of our key business drivers now are to provide information more frequently, more accurately, and by the means that consumers and the industry wish to consume that information."

Successfully addressing these requirements—the continually evolving ways that customers access and share information, and the growing importance of multiple delivery channels pressed Kelley Blue Book to improve its service offerings technologically to keep the company competitive.

Solution:

After a careful evaluation of its options, Kelley Blue Book chose to develop a digital marketing solution built on the Microsoft platform. The technology helps to expand and enhance the company's Internet-based offerings while bolstering its position as an industry leader.

Contd...

Notes

"The Microsoft technology really serves the purposes of what we wanted to do from an IT perspective, which was to build a scalable infrastructure that supports all of our applications using a common platform," says Lapin. "It helps us to deliver innovative products to our customers, improve time-to-market, and reduce our costs."

Working with Neudesic, a Microsoft Gold Certified Partner, the Kelley Blue Book IT team used Microsoft .NET Framework version 3.5 Service Pack 1, Microsoft ASP.NET, Microsoft Visual Studio 2008 Professional Edition, and Windows Presentation Foundation to enhance Internet content delivery. The Microsoft Silverlight browser plug-in was used to create the rich user interactivity. Kelley Blue Book also took advantage of the Microsoft Technology Center for an architecture design session and system prototyping.

At one point, the company briefly considered building applications in Adobe Flash but dismissed the notion because this would require it to adopt an additional programming language outside of its platform strategy. "If we were to build Flash applications, we would have to find people with additional skill sets both in Flash design and in the use of Adobe Flex," says Lapin. "By using Microsoft technologies, our designers can work in the environment that they are most familiar with."

Using the Microsoft products, Kelley Blue Book and Neudesic developed two significant enhancements to the company's online presence. The first is a mobile version of www.kbb.com that was created with technology known as the ASP.NET Model-View-Controller architectural pattern, which helps to deliver rich Web pages to mobile devices using smaller amounts of code than what is typically required for standard Web sites. The goal in rolling out an enhanced mobile offering was to increase the number of visits from mobile devices because, according to Kelley Blue Book business intelligence data, visits from mobile devices in the past only lasted for a single page view.

The company also developed its Perfect Car Finder: Photo Edition tool using Silverlight Deep Zoom, a feature of the Silverlight technology that lets viewers quickly zoom in and out of high-resolution images of autos while presenting information in an engaging, powerful way. By clicking a button inside the photo, buyers can learn about a vehicle's fuel efficiency, body style, price, customer ratings over time, and more. "Buyers can sort different vehicles according to these criteria and view the availability of different kinds of vehicles in a given geographical area, among other options," Yaros says. "And they can do all of this with ease and efficiency, thanks to the work we did with the Silverlight Deep Zoom technology."

To respond to growing customer interest around social networking integration, Kelley Blue Book now posts automobile information to social media channels, including Twitter, Facebook, and YouTube. These postings augment the company's Web site with alternative ways to find and share automobile-related news, reviews, and information.

Benefits

The decision to enhance its online presence using Microsoft products is delivering numerous benefits to Kelley Blue Book and its customers. The enhanced online offering provides customers with rich information in a compelling, interactive environment. The Web-based technology empowers consumers, dealers, and car makers to make better-informed decisions with information that is delivered quickly and efficiently across multiple channels. The Microsoft products also provide Kelley Blue Book with a powerful, scalable platform that can help the company continually modify and enhance its Web presence in a dynamic and competitive marketplace.

Contd...

Notes

Rich Information, Interactive Experience

With an abundance of choices for online information, consumers today expect Web content to be delivered in a visually engaging, interactive fashion. Using Microsoft development tools, Kelley Blue Book found the means to enhance its digital marketing efforts with this kind of content delivery experience.

"We feel that one of our coolest new features for our customers is the Perfect Car Finder," says Yaros. "We offered a similar feature on our site in the past, which let consumers look at different vehicles side-by-side and find what they wanted to see most on a vehicle. But with Silverlight, we've expanded on that. It's now completely dynamic."

With the Perfect Car Finder, viewers can dig deep into comparisons of car models and even review geographic differences that will affect what they might pay—for example, in Kansas and San Francisco—for roughly the same car.

"A consumer can very quickly sort through what they want to see most, and what is presented to them changes very quickly and very visually," says Yaros. "Our new Perfect Car Finder tool with the Microsoft Silverlight Deep Zoom technology enables us to do things we just couldn't do before in terms of delivering valuable information in a compelling way."

Better-Informed Customers

In addition to helping deliver an innovative customer experience, the Microsoft products help Kelley Blue Book deliver information to consumers faster than ever.

"Prices are changing much more quickly due to economic conditions," says Yaros.

"We have moved to producing weekly values on cars, instead of every month or two as in the past. The Microsoft platform helps us quickly deliver information to consumers and sellers so they have the latest information with which to conduct transactions."

This information is not just available to desktop-based PC users. With the Micro-soft development tools, Kelley Blue Book created a mobile version of its Web content that makes the information easily accessible on platform-agnostic mobile devices.

"We're providing consumers with a new way to get Kelley Blue Book information while they're shopping," says Yaros. "If a consumer goes into a dealership, looks at a new car, and wants to know, 'Is the price that's on this car the best price that I could get?', they can pull out their mobile device to see what Kelley Blue Book says it should be valued at, if the price is competitive, and if there are other cars available that might have a better price that meets their needs. This is very, very empowering, and it really changes the situation for the consumer—and for dealers, who do best when they know a consumer is well-informed and is entering into a transaction with confidence."

Increased Web Traffic and Ad Revenue

The rich, interactive information, combined with expansion of delivery channels, is also benefitting Kelley Blue Book by significantly increasing Web traffic, which, in turn, helps the company's Web-based advertising revenues.

Kelley Blue Book used business intelligence tools to measure its business data before and after the solution was deployed. The business data is critical for the company's business strategy of developing new service offerings and driving more interaction with customers through the company's mobile and Web sites.

Contd...

Notes

"Since introducing our new Microsoft .NET–based mobile product, we have seen Web pages-per-visit increase from 2 to 10," says Yaros. "And we've seen total page views grow from about 700,000 a month to 1.5 million a month. As a result, our advertising revenue has increased proportionately, positively impacting the bottom line."

Powerful, Flexible Platform

The platform used to build the digital marketing solutions serves as a broad, stable foundation that is helping Kelley Blue Book respond to market drivers with technology that is flexible, adaptable, and cost-effective. Lapin notes that the tight integration of Microsoft products and the wide availability of Windows development expertise have helped Kelley Blue Book enormously in its drive to stay an industry leader.

"When we decided to build on the Microsoft platform, we found that it was easy to take developers who are accustomed to Web forms programming and move them into Silverlight programming. The learning curve was very short since Silverlight and Web forms share a common development language and a common development environment," Lapin says. "The Microsoft technology really supports our organizational goals—deploying a platform that will help us build new, innovative products to support our customers, in a way that we could get to market quickly without having to spend a lot of money."

Moving forward, the company expects to expand and enhance the ways in which it delivers automotive information using the Microsoft products. "The mobile product and the new Perfect Car Finder are just the beginning for us," Yaros says. "We'll soon have other products that enable us to reach consumers, dealers, and equipment manufacturers through new devices, through new ways of sharing information, and through new ways of buying cars and selling cars. We expect to be in the middle of it. And our relationship and partnership with Microsoft is giving us the confidence to know that we can meet the future effectively."

11.10 Summary

- When you create applications that use a common header, footer, or navigation section on pretty much every page of the application, master pages are a great solution.
- Master pages are easy to implement and enable you to make changes to each and every page of your application by changing a single file.
- Imagine how much easier this makes managing large applications that contain thousands of pages.
- This unit described master pages in ASP.NET 2.0 and explained how you build and use master pages within your Web applications.
- In addition to the basics, the unit covered master page event ordering, caching, and specific master pages for specific containers.
- In the end, when you are working with templated applications, master pages should be your first option the power of this approach is immense.

11.11 Keywords

Content Page: In ASP.NET, a Web page that is configured to be merged with a master page to create a complete page.

Managed Code: Code that is executed by the CLR. Managed code provides information (i.e., metadata) to allow the CLR to locate methods encoded in assembly modules, store and retrieve security information, handle exceptions, and walk the program stack.

Notes

Master Page: In ASP.NET, a page that defines the layout for a set of pages. A master page can contain static text and controls that should appear on all pages. Master pages are merged at run time with content pages that define page-specific content.

Visual Studio 2005(VS .NET): The third version of Visual Studio .NET that was also known by the code name Whidbey. This version is due to release in 2005.

11.12 Self Assessment

Fill in the blanks:

1. In the days of Classic Active Server Pages, one popular option was to put all the common sections into what was called an ..... file.
2. Web pages sometimes displayed strange results because of inappropriate or nonexistent tag closings or .....
3. ..... live outside the pages you develop, while user controls live within your pages and are doomed to duplication.
4. Master pages use a ..... file extension, whereas content pages use the .aspx file extension.
5. Master pages can consist of just ..... content area.
6. From any content page, you can easily assign a master page .....
7. The ..... event is the earliest point in which you can access the Page lifecycle.

State whether the following statements are True or False:

8. Visual Studio 2005 corrects some of the problems by rendering user-control content in the design view.
9. You can add any content to these defined content areas as if you were working with a regular .aspx page.
10. If you declare your master page in the Web.config file, you can create only one content page that uses this master page.

11.13 Review Questions

1. What's the most effective way to apply styles on your web page?
2. If a style sheet has rules applying to the text of the whole page, but one specific paragraph has a different style rule applied, which takes precedence?
3. What two commands can you use to apply a style sheet to your page, and where do you place them?
4. What is the purpose of a master page?
5. How many different master pages can you associate with a particular content page?
6. When you are trying to change the content of a master page at runtime, what does the content page need to affect the master page?
7. What method could you use in a Button Click event handler to navigate to another page?
8. What file do you need for all the navigation controls to work? How is this file generated?
9. What control do you use to enable the navigation controls to access the file?
10. What do you have to do to connect the SiteMapPath control to a data source?

Notes

Answers: Self Assessment

1.	include	2.	openings
3.	Master pages	4.	.master
5.	one	6.	programmatically
7.	Page_PreInit	8.	True
9.	True	10.	False

11.14 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Unit 12: ADO.NET and Data Binding

Notes

CONTENTS

Objectives

Introduction

12.1 Why ADO.NET?

12.1.1 A Brief History

12.1.2 ADO.NET isn't a New Version of ADO

12.2 ADO.NET Architecture

12.3 Configuring your Database

12.3.1 SQL Server Express

12.3.2 Browsing and Modifying Databases in Visual Studio

12.3.3 The sqlcmd Command-Line Tool

12.4 ADO.NET Basics

12.4.1 Basic ADO.NET Namespaces and Classes:

12.4.2 Using the Connection Object

12.4.3 Using the Command Object

12.4.4 Using the DataReader Object

12.4.5 Using Data Adapter

12.4.6 Using Parameters

12.4.7 Understanding DataSet and DataTable

12.5 Direct Data Access

12.5.1 Creating a Connection

12.5.2 The Select Command

12.5.3 The DataReader

12.6 Single-value Data Binding

12.6.1 A Simple Data Binding Example

12.6.2 Simple Data Binding with Properties

12.6.3 Problems with Single-value Data Binding

12.7 Repeated-value Data Binding

12.7.1 Data Binding with Simple List Controls

12.7.2 A Simple List Binding Example

Contd...

Notes

12.7.3 Strongly Typed Collections
12.7.4 Multiple Binding
12.8 Summary
12.9 Keywords
12.10 Self Assessment
12.11 Review Questions
12.12 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe configuring your database
- Know ADO.NET basics
- Explain direct data access
- Describe single-value data binding
- Explain repeated-value data binding

Introduction

The ADO.NET functionality is different from what you are used to with ADO. Although similar in concept, the ADO.NET objects follow a disconnected paradigm. One of the major differences is that you are not necessarily retrieving the equivalent of a recordset, but an entire schema structure.

This unit provides information on programming with data management features that are part of ADO.NET. The discussion begins with the basics of ADO.NET and later dives into the ways you can use the newly added advanced ADO.NET features to manage data contained in a relational database.

ADO.NET was first introduced in version 1.0 of the .NET Framework and provided an extensive array of features to handle data either live while connected to the database or when disconnected. With the introduction of ADO.NET 2.0, the already-extensive features list has grown even larger. Some of the newly added features include the capability to bulk load large quantities of data from a variety of sources, to batch process updates to the database with fewer round trips back to the database server, to reuse the same live connection for multiple operations, as well as to achieve asynchronous access to the database.

12.1 Why ADO.NET?

Almost all applications require some data access. Before the .NET Framework, developers used data access technologies such as ODBC, OLE DB, and ADO in their applications; with the introduction of .NET, Microsoft created a new way to work with data, called ADO.NET. This is the only technology you need to use in .NET to access data sources, but in reality several of its predecessors are still lurking under the hood. We'll start this unit by telling you what they are and how they fit into the grand scheme of things.

Notes

12.1.1 A Brief History

Back at the end of the 1980s, several big database servers and a few small ones were available to buy and use. Each used SQL (with one or two of their own proprietary extensions thrown in for good measure) to access and manipulate the data they contained, and each had their own proprietary set of programming interfaces for developers to tie their own applications into the servers. This forced a great deal of extra work onto developers, as in order to support more than one database, they would have to rewrite all their application's data access code for each database. The database vendors saw the problem and collaborated to create a common set of (really) low-level interfaces that all their servers would support. They called this API set Open DataBase Connectivity (ODBC).

However, writing ODBC code directly is tricky. Microsoft saw an opportunity to make things easier and wrote a set of Component Object Model (COM) components that was much easier to develop with and also let applications access data that wasn't stored in a database something you couldn't do with ODBC. Data stored in text files, spreadsheets, and other sources could now be used as well. This new technology, called OLE DB, used ODBC to access old databases where necessary and its own code to access databases and other data sources. Based on its rapid uptake, Microsoft decided to use it as the cornerstone for its Universal Data Access (UDA) strategy.

One of the aims of UDA was to provide a more object-oriented interface to data access than the somewhat procedural one that ODBC and OLE DB presented. Thus, Data Access Objects (DAO) and Remote Data Objects (RDO) were born in 1997, and a year later ActiveX Data Objects (ADO 2.0) gave classic ASP applications access to data sources in a constantly connected environment. ADO was built on top of OLE DB, which means that an application can use ADO, which talks to OLE DB, which talks to ODBC, which talks to a database, if you're using older databases such as Paradox or dBASE. ADO was even simpler than OLE DB to use, however.

With the multilayered data access model and the connected nature of ADO, you could easily end up sapping server resources and creating a performance bottleneck. ADO served well, but ADO.NET is much, much better.

12.1.2 ADO.NET isn't a New Version of ADO

ADO.NET is a completely new data access technology, with a new design that was built entirely from scratch. Let's first get this cleared up: ADO.NET doesn't stand for ActiveX Data Objects .NET. Why? It's because of many reasons, but the following are the two most important ones:

1. ADO.NET is an integral part of the .NET Framework, not an external entity.
2. ADO.NET isn't a collection of ActiveX components.

The name ADO.NET is analogous to ADO because Microsoft wanted developers to feel at home using ADO.NET and didn't want them to think they'd need to "learn it all over again," so it purposely named and designed ADO.NET to offer similar features implemented in a different way.

During the .NET Framework design, Microsoft realized that ADO wasn't going to fit in. ADO was available as an external package based on COM objects, requiring .NET applications to explicitly include a reference to it. In contrast, .NET applications are designed to share a single model, where all libraries are integrated into a single framework, organized into logical namespaces, and declared public to any application that wants to use them. It was wisely decided that the .NET data access technology should comply with the .NET architectural model. Hence, ADO.NET was born.

ADO.NET is designed to accommodate both connected and disconnected environments.

Notes

Also, ADO.NET embraces the fundamentally important Extensible Markup Language (XML) standard, much more than ADO did, since the explosion in XML use came about after ADO was developed. With ADO.NET you cannot only use XML to transfer data between applications, but you can also export data from your application into an XML file, store it locally on your system, and retrieve it later when you need to do so.

Performance usually comes with a price, but in the case of ADO.NET, the price is definitely reasonable. Unlike ADO, ADO.NET doesn't transparently wrap OLE DB providers; instead, it uses managed data providers that are designed specifically for each type of data source, thus leveraging their true power and adding to overall application speed and performance. It's only if the data source doesn't have its own native data provider that you must fall back to using a generic OLE DB or ODBC data provider.

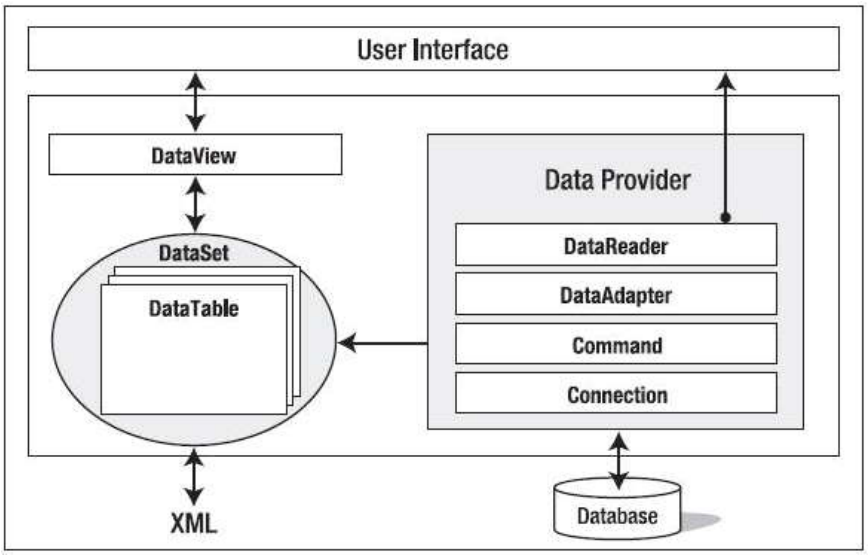
ADO.NET also works in both connected and disconnected environments. You can connect to a database, remain connected while simply reading data, and then close your connection, which is a process similar to ADO. Where ADO.NET really begins to shine is in the disconnected world. If you need to edit database data, maintaining a continuous connection would be costly on the server. ADO.NET gets around this by providing a sophisticated disconnected model. Data is sent from the server and cached locally on the client. When you're ready to update the database, you can send the changed data back to the server, where updates and conflicts are managed for you.

In ADO.NET, when you retrieve data, you use an object known as a data reader. When you work with disconnected data, the data is cached locally in a relational data structure called a dataset.

12.2 ADO.NET Architecture

Figure 12.1 presents the most important architectural features of ADO.NET.

Figure 12.1: ADO.NET Architecture



ADO.NET has two central components: data providers and datasets.

A data provider connects to a data source and supports data access and manipulation.

A dataset supports disconnected, independent caching of data in a relational fashion, updating the data source as required. A dataset contains one or more data tables. A data table is a row-and-column representation that provides much the same logical view as a SQL table.

Notes

Example: You can store the data from the Northwind database's Employees table in a data table and manipulate the data as needed.

In Figure 12.2, notice the DataView class (in the System.Data namespace). This isn't a data provider component. Data views are used primarily to bind data to Windows and Web Forms.

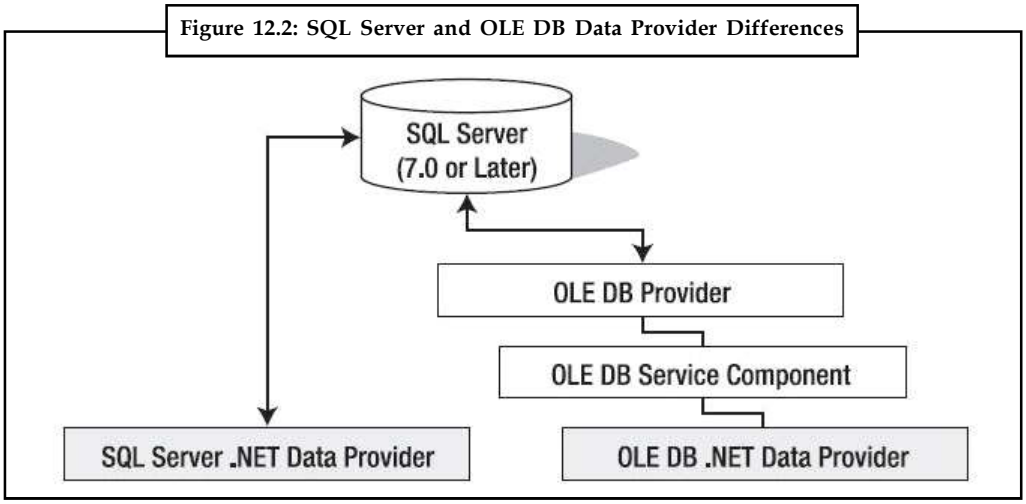
As you saw in Table 12.1, each data provider has its own namespace. In fact, each data provider is essentially an implementation of interfaces in the System.Data namespace, specialized for a specific type of data source.

Example: If you use SQL Server version 7 or newer (SQL Server 2000 is actually version 8) as your database manager, you should use the SQL Server data provider (System.Data.SqlClient) because it's the most efficient way to work with these versions. This data provider communicates natively with SQL Server, bypassing the layers that OLE DB and ODBC connections have to use.

The OLE DB data provider supports access to older versions of SQL Server as well as to other databases, such as Access, DB2, MySQL, and Oracle. However, native data providers (such as System.Data.OracleClient) are preferable for performance, since the OLE DB data provider works through two other layers (the OLE DB service component and the OLE DB provider) before reaching the data source.

Figure 12.2 illustrates the difference between using the SQL Server and OLE DB data providers to access a SQL Server (version 7.0 or higher) database.

Figure 12.2: SQL Server and OLE DB Data Provider Differences



If your application connects to an older version of SQL Server (6.5 or newer) or to more than one kind of database server at the same time (for example, an Access and an Oracle database connected simultaneously), only then should you choose to use the OLE DB data provider.

No hard-and-fast rules exist; you can use both the OLE DB data provider for SQL Server and the Oracle data provider (System.Data.OracleClient) if you want, but it's important you choose the best provider for your purpose. Given the performance benefits of the server-specific data providers, if you use SQL Server or MSDE, 99 percent of the time you should be using the System.Data.SqlClient classes.

Notes

Before you look at what each kind of data provider does and how it's used, you need to be clear on their core functionality. Each .NET data provider is designed to do the following two things very well:

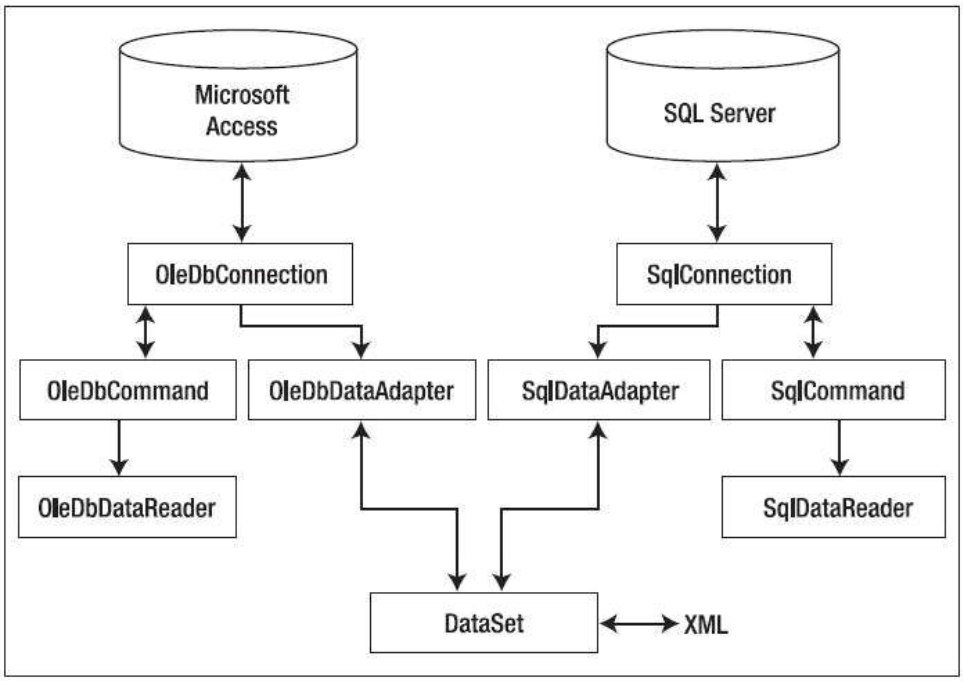
1. Provide access to data with an active connection to the data source.
2. Provide data transmission to and from the independent datasets through data adapters.

Database connections are established by using the data provider's connection class (for example, System.Data.SqlClient.SqlConnection). Other components such as data readers, commands, and data adapters support retrieving data, executing SQL statements, and reading or writing to datasets, respectively.

As you've seen, each data provider is prefixed with the type of data source it connects to (for instance, the SQL Server data provider is prefixed with Sql), so its connection class is named SqlConnection. The OLE DB data provider's connection class is named OleDbConnection.

Look now at the object model of these two providers in Figure 12.3. You'll work simultaneously with different databases (Access and SQL Server) so you can see how easy it is to switch between data providers and what, if any, their main visible differences are. Note that the two data providers support the same application and share a single dataset.

Figure 12.3: ADO.NET concurrently using Access and SQL Server



The OLE DB data provider belongs to the System.Data.OleDb namespace; the SQL Server data provider belongs to System.Data.SqlClient. Both data providers seem to have a similar architecture, though they're actually very different internally.

Using the SQL Server Data Provider

The .NET data provider for SQL Server (7.0 or newer) is located in the System.Data.SqlClient namespace. This data provider communicates directly with the server using its native network protocol instead of through a number of layers (the way OLE DB does).

Notes

Table 12.1 describes some important classes in the SqlClient namespace.

Table 12.1: Commonly used SqlClient Classes

Classes	Description
SqlCommand	Executes SQL queries, statements, or stored procedures
SqlConnection	Represents a connection to a SQL Server database
SqlDataAdapter	Represents a bridge between a dataset and a data source
SqlDataReader	Provides a forward-only, read-only data stream of the results
SqlError	Holds information on SQL Server errors and warnings
SqlParameter	Represents a command parameter
SqlTransaction	Represents a SQL Server transaction

Another namespace, System.Data.SqlTypes, maps SQL Server data types to .NET types, both enhancing performance and making developers' lives a lot easier.

Task ADO provides a connection object to set up and maintain a connection between the application and the ODBC data source. How it possible? Discuss

12.3 Configuring your Database

Before you can run any data access code, you need a database server to take your command. Although there are dozens of good options, all of which work equally well with ADO.NET (and require essentially the same code), a significant majority of ASP.NET applications use Microsoft SQL Server.

This unit includes code that works with SQL Server 7 or later, although you can easily adapt the code to work with other database products. Ideally you'll use SQL Server 2005 (with Service Pack 2) or SQL Server 2008. Microsoft is phasing out older versions, and they don't have support for Windows Vista and Windows Server 2008.

If you don't have a full version of SQL Server, there's no need to worry you can simply install the free SQL Server Express Edition (as described in the next section). It includes all the database features you need to develop and test a web application.

Notes This unit (and the following two units) use examples drawn from the pubs and Northwind databases, which are sample databases included with some versions of Microsoft SQL Server. These databases aren't preinstalled in all versions of SQL Server, and they're noticeably absent from SQL Server 2005. However, you can easily install them using the scripts provided with the online samples. See the readme.txt file for full instructions.

12.3.1 SQL Server Express

If you don't have a test database server handy, you may want to use SQL Server 2005 Express Edition, the free data engine included with some versions of Visual Studio and downloadable separately.

SQL Server Express is a scaled-down version of SQL Server 2005 that's free to distribute. SQL Server Express has certain limitations for example, it can use only one CPU and a maximum of

Notes

1GB of RAM; databases can't be larger than 4GB; and graphical tools aren't included. However, it's still remarkably powerful and suitable for many midscale websites. Even better, you can easily upgrade from SQL Server Express to a paid version of SQL Server if you need more features later.

12.3.2 Browsing and Modifying Databases in Visual Studio

As an ASP.NET developer, you may have the responsibility of creating the database required for a web application. Alternatively, it may already exist, or it may be the responsibility of a dedicated database administrator. If you're using a full version of SQL Server, you'll probably use a graphical tool such as SQL Server Management Studio to create and manage your databases.

If you don't have a suitable tool for managing your database, or you don't want to leave the comfort of Visual Studio, you can perform many of the same tasks using Visual Studio's Server Explorer window.

Here's how you can get started. First, choose View → Server Explorer from the Visual Studio menu to show the Server Explorer window. Then, using the Data Connections node in the Server Explorer, you can connect to existing databases or create new ones. Assuming you've installed the pubs database (see the readme.txt file for instructions), you can create a connection to it by following these steps:

1. Right-click the Data Connections node, and choose Add Connection. If the Choose Data Source window appears, select Microsoft SQL Server and then click Continue.
2. If you're using a full version of SQL Server, enter localhost as your server name. This indicates the database server is the default instance on the local computer. (Replace this with the name of a remote computer if needed.) If you're using SQL Server Express, you'll need to use the server name localhost\SQLEXPRESS instead, as shown in Figure 12.4. The SQLEXPRESS part indicates that you're connecting to a named instance of SQL Server. By default, this is the way that SQL Server Express configures itself when you first install it.

Figure 12.4: Creating a Connection in Visual Studio

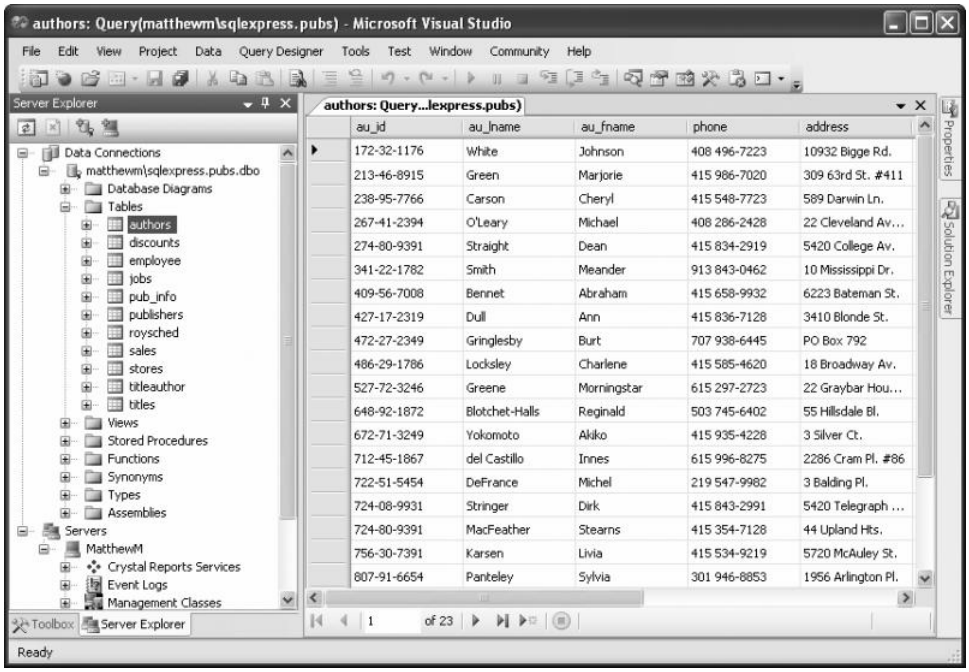
Notes

3. Click Test Connection to verify that this is the location of your database. If you haven't installed a database product yet, this step will fail. Otherwise, you'll know that your database server is installed and running.

4. In the Select or Enter a Database Name list, choose the pubs database. (In order for this to work, the pubs database must already be installed. You can install it using the database script that's included with the sample code, as explained in the following section.) If you want to see more than one database in Visual Studio, you'll need to add more than one data connection.

5. Click OK. The database connection will appear in the Server Explorer window. You can now explore its groups to see and edit tables, stored procedures, and more. For example, if you right-click a table and choose Show Table Data, you'll see a grid of records that you can browse and edit, as shown in Figure 12.5.

Figure 12.5: Editing Table Data in Visual Studio



12.3.3 The sqlcmd Command-Line Tool

SQL Server 2005 (and 2008) include a handy command-line tool named sqlcmd.exe that you can use to perform database tasks from a Windows command prompt. Compared to a management tool like SQL Server Management Studio, sqlcmd doesn't offer many frills. It's just a quick-and-dirty way to perform a database task. Often, sqlcmd is used in a batch file for example, to create database tables as part of an automated setup process.

The sqlcmd tool is installed as part of SQL Server 2005 (and 2008), and it's found in a directory like c:\Program Files\Microsoft SQL Server\90\Tools\Binn. The easiest way to run sqlcmd is to launch the Visual Studio command prompt (open the Start menu and choose Programs → Microsoft Visual Studio 2008 → Visual Studio Tools → Visual Studio 2008 Command Prompt). This opens a command window that has the SQL Server directory set in the path variable. As a result, you can use sqlcmd anywhere you want, without typing its full directory path.

Notes

When running sqlcmd, it's up to you to supply the right parameters. To see all the possible parameters, type this command:

```
sqlcmd -?
```

Two commonly used sqlcmd parameters are –S (which specifies the location of your database server) and –i (which supplies a script file with SQL commands that you want to run). For example, the downloadable code samples include a file named InstPubs.sql that contains the commands you need to create the pubs database and fill it with sample data. If you're using SQL Server Express, you can run the InstPubs.sql script using this command:

```
sqlcmd -S localhost\SQLEXPRESS -i InstPubs.
```

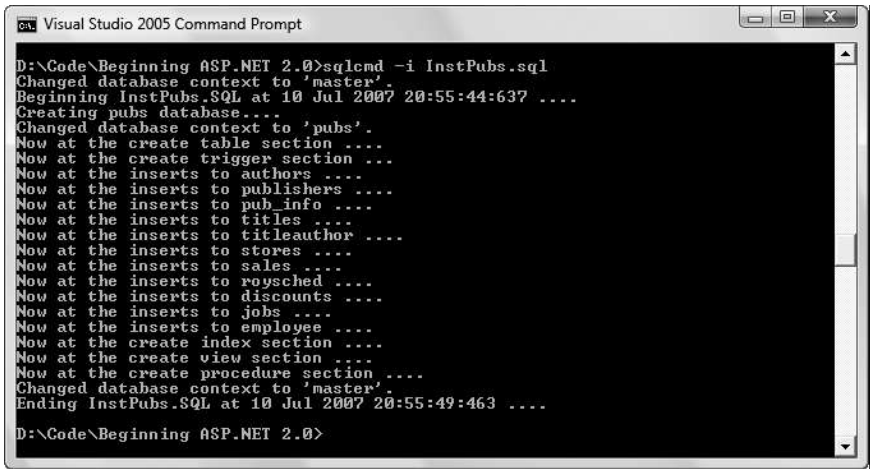
If you're using a full version of SQL Server on the local computer, you don't need to supply the server name at all:

```
sqlcmd -i InstPubs.sql
```

And if your database is on another computer, you need to supply that computer's name with the –S parameter (or just run sqlcmd on that computer).

Figure 12.6 shows the feedback you'll get when you run InstPubs.sql with sqlcmd.

Figure 12.6: Running a SQL Script with sqlcmd.exe



12.4 ADO.NET Basics

12.4.1 Basic ADO.NET Namespaces and Classes

The six basic ADO.NET namespaces are shown in the following table. In addition to these namespaces, each new data provider can have its own namespace. As an example, the Oracle .NET data provider adds a namespace of Microsoft.Data.OracleClient.

ADO.NET has three distinct types of classes commonly referred to as Disconnected, Shared, and Data Providers. The Disconnected classes provide the basic structure for the ADO.NET framework. A good example of this type of class is the DataTable class. The objects of this class are capable of storing data without any dependency on a specific data provider. The Shared classes form the base classes for data providers and are shared among all data providers. The Data Provider classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases. The SqlClient data provider, for example, works only with the SQL Server database.

Notes

Namespace	Description
System.Data	This namespace is the core of ADO.NET. It contains classes used by all data providers. It contains classes to represent tables, columns, rows, and the DataSet. It also contains several very useful interfaces, such as IDbCommand, IDbConnection, and IDbDataAdapter. These interfaces are used by all managed providers, enabling them to plug into the core of ADO.NET.
System.Data.Common	This namespace defines common classes that are used as base classes for data providers. All data providers share these classes. A few examples are DbConnection and DbDataAdapter.
System.Data.OleDb	This namespace defines classes that work with OLE-DB data sources using the .NET OleDb data provider. It contains classes such as OleDbConnection and OleDbCommand.
System.Data.Odbc	This namespace defines classes that work with the ODBC data sources using the .NET ODBC data provider. It contains classes such as OdbcConnection and OdbcCommand.
System.Data.SqlClient	This namespace defines a data provider for the SQL Server 7.0 or higher database. It contains classes such as SqlConnection and SqlCommand.
System.Data.SqlTypes	This namespace defines a few classes that represent specific data types for the SQL Server database.

A data provider contains Connection, Command, DataAdapter, and DataReader objects. In a typical ADO.NET programming, you first create the Connection object and provide it with the necessary information, such as the connection string. You then create a Command object and provide it with the details of the SQL command that is to be executed. This command can be an inline SQL text command, a stored procedure, or direct table access. You can also provide parameters to these commands if needed. After you create the Connection and the Command objects, you must decide whether the command returns a result set. If the command doesn't return a result set, you can simply execute the command by calling one of its several Execute methods. On the other hand, if the command returns a result set, you must make a decision about whether you want to retain the result set for future uses without maintaining the connection to the database. If you want to retain the result set, you must create a DataAdapter object and use it to fill a DataSet or a DataTable object. These objects are capable of maintaining their information in a disconnected mode. However, if you don't want to retain the result set, but rather to simply process the command in a swift fashion, you can use the Command object to create a DataReader object. The DataReader object needs a live connection to the database, and it works as a forward-only, read-only cursor.

12.4.2 Using the Connection Object

The Connection object creates a link to the data source. This object needs the necessary information to discover the data source and to log in to it properly. This information is provided via a connection string.

The properties for the SqlConnection class are shown in the following table. The SqlConnection is a class that is specific to the SqlClient provider. As discussed earlier in this unit, the SqlClient provider is built for working with the SQL Server 7.0 and higher databases.

Property	Description
Datasource	This read-only property returns the name of the instance of the SQL Server database used by the SqlConnection object.
Database	This read-only property returns the name of the database to use after the connection is opened.
State	This read-only property returns the current state of the connection. The possible values are Broken, Closed, Connecting, Executing, Fetching, and Open.
ConnectionString	This property allows you to read or provide the connection string that should be used by the SqlConnection object.

Notes

12.4.3 Using the Command Object

The Command object uses the Connection object to execute SQL queries. These queries can be in the form of inline text, a stored procedure, or direct table access. If the SQL query uses the SELECT clause, the result set it returns is usually stored in either a DataSet or a DataReader object. The command object provides a number of Execute methods that can be used to perform the SQL queries in a variety of fashions.

First take a look at some useful properties of the SqlCommand class, as shown in the following table.

Property	Description
CommandText	This read-write property allows you to set or retrieve either the T-SQL statement or the stored procedure name.
CommandTimeout	This read-write property gets or sets the number of seconds to wait while attempting to execute a command. The command is aborted after it times out and an exception is thrown. The default is 30 seconds.
CommandType	This read-write property indicates the way the CommandText property should be interpreted. The possible values are StoredProcedure, TableDirect, and Text.
Connection	This read-write property gets or sets the SqlConnection object that should be used by this command object.

Now look at the various Execute methods that can be called on a Command object.

Property	Description
ExecuteNonQuery	This method executes the command and returns the number of rows affected.
ExecuteReader	This method executes the command and returns an object of SqlDataReader class. The data reader is a read-only and forward-only cursor.
ExecuteRow	This method executes the command and returns an object of the SqlRecord class. This object contains a single returned row.
ExecuteScalar	This method executes the command and returns the first column of the first row in the form of a generic object. The remaining rows and columns are ignored.
ExecuteXmlReader	This method executes the command and returns an object of the XmlReader class. This method enables you to use a command that returns the results set in the form of an XML document.

12.4.4 Using the DataReader Object

The DataReader object is a simple forward-only and read-only cursor. It requires a live connection with the data source and provides a very efficient way of looping and consuming all or part of the result set. This object cannot be directly instantiated. Instead, you must call the ExecuteReader method of the Command object to obtain a valid DataReader object. Be sure to close the connection when you are done using the data reader. Otherwise, the connection stays alive until it is explicitly closed. You can close the connection after using the data reader in one of two ways. One way is to provide the CommandBehavior.CloseConnection enumeration while calling the ExecuteMethod of the Command object. This approach works only if you loop through the data reader until you reach the end of file, at which point the reader object automatically closes the connection for you. However, if you don't want to keep reading the data reader until the end of file, you can call the Close method of the Connection object yourself.

Listing 12.1 shows the Connection, Command, and DataReader objects in action. It shows how to connect with the Northwind database, read the Customers table, and show the results in a GridView control.

Listing 12.1: The SqlConnection, SqlCommand, and SqlDataReader objects

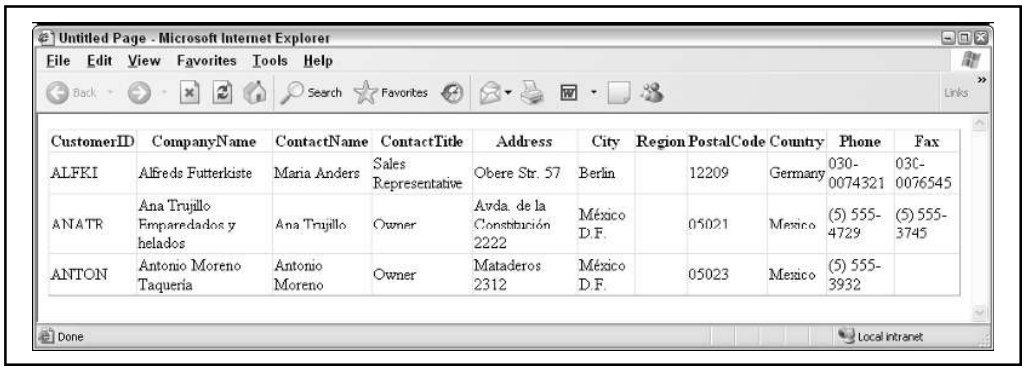
Notes

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
Protected Sub Page_Load(ByVal sender As Object, _
ByVal e As System.EventArgs)
If Not Page.IsPostBack Then
Dim MyConnection As SqlConnection
Dim MyCommand As SqlCommand
Dim MyReader As SqlDataReader
MyConnection = New SqlConnection()
MyConnection.ConnectionString = _
ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString
MyCommand = New SqlCommand()
MyCommand.CommandText = "SELECT TOP 3 * FROM CUSTOMERS"
MyCommand.CommandType = CommandType.Text
MyCommand.Connection = MyConnection
MyCommand.Connection.Open()
MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection)
gvCustomers.DataSource = MyReader
gvCustomers.DataBind()
MyCommand.Dispose()
MyConnection.Dispose()
End If
End Sub
</script>
<html>
<body>
<form id="form1" runat="server">
<div>
<asp:GridView ID="gvCustomers" runat="server">
</asp:GridView>
</div>
</form>
</body>
</html>
```

The code shown in Listing 12.1 uses the SqlConnection class to create a connection with the Northwind database using the connection string stored in the Web.config file. It then creates a Command object using the SqlCommand class and provides it with command text, command type, and connection properties. After the command and the connection are created, the code

Notes

opens the connection and executes the command by calling the ExecuteReader method of the MyCommand object. After receiving the data reader from the Command object, you simply data bind it to a GridView control. The result of the code is shown in figure.



CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	

12.4.5 Using Data Adapter

The SqlDataAdapter is a special class whose purpose is to bridge the gap between the disconnected DataTable objects and the physical data source. The SqlDataAdapter provides a two-day data transfer mechanism. It is capable of executing a SELECT statement on a data source and transferring the result set into a DataTable object. It is also capable of executing INSERT, UPDATE, and DELETE statements and extracting the input data from a DataTable object.

The commonly used properties offered by the SqlDataAdapter class are shown in the following table.

Property	Description
SelectCommand	This read-write property sets or gets an object of type SqlCommand. This command is automatically executed to fill a DataTable with the result set.
InsertCommand	This read-write property sets or gets an object of type SqlCommand. This command is automatically executed to insert a new record to the SQL Server database.
UpdateCommand	This read-write property sets or gets an object of type SqlCommand. This command is automatically executed to update an existing record on the SQL Server database.
DeleteCommand	This read-write property sets or gets an object of type SqlCommand. This command is automatically executed to delete an existing record on the SQL Server database.

The SqlDataAdapter class also provides a method called Fill. Calling the Fill method automatically executes the command provided in the SelectCommand property, receives the result set, and copies it to a DataTable object.

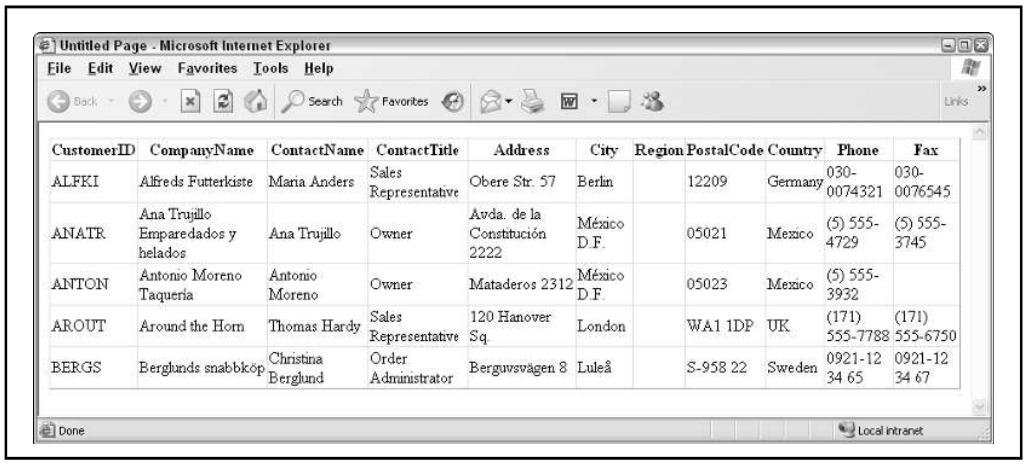
The code example in Listing 12.2 illustrates how to use an object of SqlDataAdapter class to fill a DataTable object.

Listing 12.2: Using an object of SqlDataAdapter to fill a DataTable

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>
<script runat="server">
Protected Sub Page_Load(ByVal sender As Object, _
```

Notes

```
ByVal e As System.EventArgs)

If Not Page.IsPostBack Then
Dim MyConnection As SqlConnection
Dim MyCommand As SqlCommand
Dim MyAdapter As SqlDataAdapter
Dim MyTable As DataTable = New DataTable()

MyConnection = New SqlConnection()
MyConnection.ConnectionString = _
ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString

MyCommand = New SqlCommand()
MyCommand.CommandText = " SELECT TOP 5 * FROM CUSTOMERS "
MyCommand.CommandType = CommandType.Text
MyCommand.Connection = MyConnection

MyAdapter = New SqlDataAdapter()
MyAdapter.SelectCommand = MyCommand
MyAdapter.Fill(MyTable)

gvCustomers.DataSource = MyTable.DefaultView
gvCustomers.DataBind()

MyAdapter.Dispose()
MyCommand.Dispose()
MyConnection.Dispose()

End If
End Sub
</script>
```



CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67

The code shown in Listing 12.2 creates a Connection and Command object and then proceeds to create an object of the SqlDataAdapter class. It then fills the SelectCommand property of the DataAdapter object with the Command object it had previously created. After the DataAdapter

Notes

object is ready for executing, the code executes the Fill method, passing it an object of the DataTable class. The Fill method returns a populated DataTable object. figure shows the result of executing this code.

12.4.6 Using Parameters

Most serious database programming, regardless of how simple it might be, requires you to configure SQL statements using parameters. Obviously, a discussion on the basics of ADO.NET programming is not complete without covering the use of parameterized SQL statements.

Creating a parameter is as simple as declaring an object of the SqlParameter class and providing it the necessary information, such as parameter name, type, size, direction, and so on. The following table shows the properties of the SqlParameter class.

Property	Description
ParameterName	This read-write property gets or sets the name of the parameter.
SqlDbType	This read-write property gets or sets the SQL Server database type of the parameter value.
Size	This read-write property sets or gets the size of the parameter value.
Direction	This read-write property sets or gets the direction of the parameter, such as Input, Output, or Both
SourceColumn	This read-write property maps a column from a DataTable to the parameter. It enables you to execute multiple commands using the SqlDataAdapter object and pick the correct parameter value from a DataTable column during the command execution.
Value	This read-write property sets or gets the value provided to the parameter object. This value is passed to the parameter defined in the command during runtime.

Listing 12.3 modifies the code shown in Listing 12.1 to use two parameters while retrieving the list of customers from the database.

Listing 12.3: The use of a parameterized SQL statement

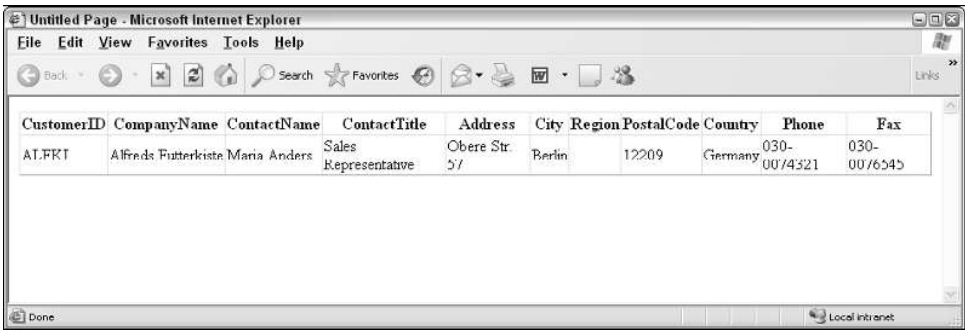
```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
Protected Sub Page_Load(ByVal sender As Object, _
ByVal e As System.EventArgs)
If Not Page.IsPostBack Then
Dim MyConnection As SqlConnection
Dim MyCommand As SqlCommand
Dim MyReader As SqlDataReader
Dim CityParam As SqlParameter
Dim ContactParam As SqlParameter

MyConnection = New SqlConnection()
MyConnection.ConnectionString = _
```

Notes

```
ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString

MyCommand = New SqlCommand()
MyCommand.CommandText = _
" SELECT * FROM CUSTOMERS WHERE CITY = @CITY AND CONTACTNAME = @CONTACT "
MyCommand.CommandType = CommandType.Text
MyCommand.Connection = MyConnection

CityParam = New SqlParameter()
CityParam.ParameterName = "@CITY"
CityParam.SqlDbType = SqlDbType.VarChar
CityParam.Size = 15
CityParam.Direction = ParameterDirection.Input
CityParam.Value = "Berlin"

ContactParam = New SqlParameter()
ContactParam.ParameterName = "@CONTACT"
ContactParam.SqlDbType = SqlDbType.VarChar
ContactParam.Size = 15
ContactParam.Direction = ParameterDirection.Input
ContactParam.Value = "Maria Anders"

MyCommand.Parameters.Add(CityParam)
MyCommand.Parameters.Add(ContactParam)

MyCommand.Connection.Open()
MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection)

gvCustomers.DataSource = MyReader
gvCustomers.DataBind()

MyCommand.Dispose()
MyConnection.Dispose()

End If

End Sub
</script>
```

The code shown in Listing 12.3 uses a parameterized SQL statement that receives the name of the city and the contact person to narrow the result set. These parameters are provided by declaring the objects of SqlParameter class and filling in the name, type, size, direction, and value properties for each object of SqlParameter class. You then add the parameters to the Command object by calling the Add method of the Parameters collection. The result of executing this code is shown in figure.

Notes



CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545

Task Discuss the connection to the database is closed using the Open method.

12.4.7 Understanding DataSet and DataTable

Most programmers agree that the DataSet class is the most commonly used part of ADO.NET in realworld, database-driven applications. This class provides mechanisms for managing data when it is disconnected from the data source. This capability to handle data in a disconnected state was first introduced in .NET during the 1.0 version of ADO.NET. The current 2.0 version of ADO.NET retains all the features of its predecessors and provides a few newer, much-needed features.

An object created from the DataSet class works as a container for other objects that are created from the DataTable class. The DataTable object represents a logical table in memory. It contains rows, columns, primary keys, constraints, and relations with other DataTable objects. Most of the disconnected datadriven programming is actually done using one or more DataTable objects. However, the previous versions of ADO.NET didn't allow you to work directly with the DataTable object for some very important tasks, such as reading and writing data to and from an XML file. It didn't even allow you to serialize the DataTable object independently. This limitation required you to always use the DataSet object to perform any operation on a DataTable. The current version of ADO.NET removes this limitation and enables you to work directly with the DataTable for all your needs. In fact, we recommend that you don't use the DataSet object unless you need to work with multiple DataTable objects and need a container object to manage them.

The current version of ADO.NET provides the capability to load a DataTable in memory by consuming a data source using a DataReader. In the past, you were sometimes restricted to creating multiple overloads of the same method just to work with both the DataReader and the DataTable objects. Now you have the flexibility to write the data access code one time and reuse the DataReader either directly or to fill a DataTable, as shown in Listing 12.4.

Listing 12.4: How to load a DataTable from a DataReader

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>
```

Notes

```
<script runat="server">
Protected Sub Page_Load(ByVal sender As Object, _
ByVal e As System.EventArgs)

If Not Page.IsPostBack Then
Dim MyConnection As SqlConnection
Dim MyCommand As SqlCommand
Dim MyDataTable As DataTable
Dim MyReader As SqlDataReader
Dim CityParam As SqlParameter

MyConnection = New SqlConnection()
MyConnection.ConnectionString = _
ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString

MyCommand = New SqlCommand()
MyCommand.CommandText = _
" SELECT * FROM CUSTOMERS WHERE CITY = @CITY "
MyCommand.CommandType = CommandType.Text
MyCommand.Connection = MyConnection

CityParam = New SqlParameter()
CityParam.ParameterName = "@CITY"
CityParam.SqlDbType = SqlDbType.VarChar
CityParam.Size = 15
CityParam.Direction = ParameterDirection.Input
CityParam.Value = "London"

MyCommand.Parameters.Add(CityParam)

MyCommand.Connection.Open()
MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection)

MyDataTable = New DataTable()

' Loading DataTable using a DataReader
MyDataTable.Load(MyReader)

gvCustomers.DataSource = MyDataTable
gvCustomers.DataBind()

MyDataTable.Dispose()
MyCommand.Dispose()
MyConnection.Dispose()
```

Notes

```
End If

End Sub
</script>
```

Not only can you load a DataTable object from a DataReader object, you can also retrieve a DataTableReader (new to the .NET Framework 2.0) from an existing DataTable object. This is accomplished by calling the CreateDataReader method of the DataTable class. This method returns an instance of the DataTableReader object that can be passed to any method that expects to receive a DataReader.

12.5 Direct Data Access

The easiest way to interact with a database is to use direct data access. When you use direct data access, you're in charge of building a SQL command (like the ones you considered earlier in this unit) and executing it. You use commands to query, insert, update, and delete information.

When you query data with direct data access, you don't keep a copy of the information in memory. Instead, you work with it for a brief period of time while the database connection is open, and then close the connection as soon as possible. This is different than disconnected data access, where you keep a copy of the data in the DataSet object so you can work with it after the database connection has been closed.

The direct data model is well suited to ASP.NET web pages, which don't need to keep a copy of their data in memory for long periods of time. Remember, an ASP.NET web page is loaded when the page is requested and shut down as soon as the response is returned to the user. That means a page typically has a lifetime of only a few seconds (if that).

Notes Although ASP.NET web pages don't need to store data in memory for ordinary data management tasks, they just might use this technique to optimize performance. For example, you could get the product catalog from a database once, and keep that data in memory on the web server so you can reuse it when someone else requests the same page.

To query information with simple data access, follow these steps:

1. Create Connection, Command, and DataReader objects.
2. Use the DataReader to retrieve information from the database, and display it in a control on a web form.
3. Close your connection.
4. Send the page to the user. At this point, the information your user sees and the information in the database no longer have any connection, and all the ADO.NET objects have been destroyed.

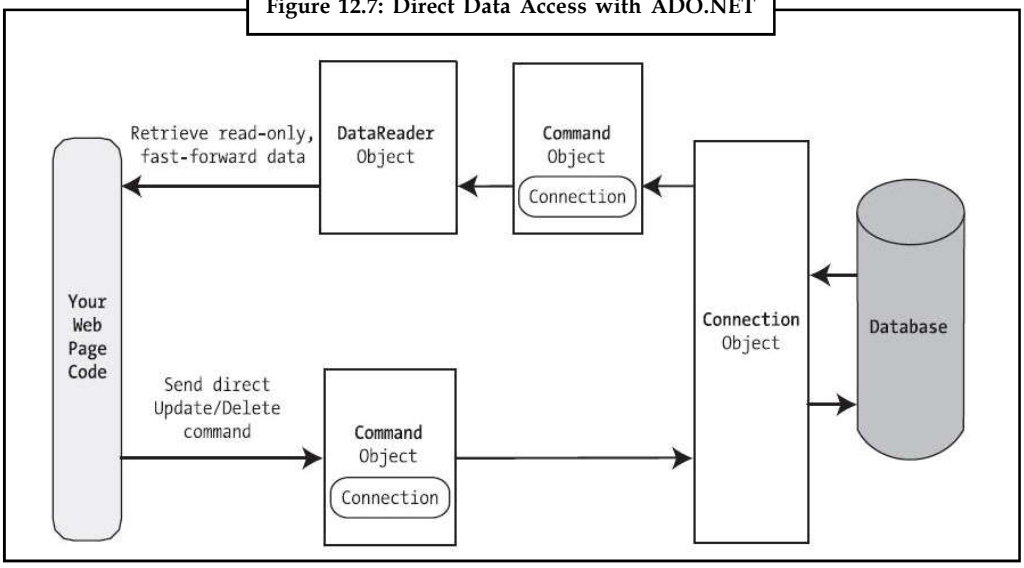
To add or update information, follow these steps:

1. Create new Connection and Command objects.
2. Execute the Command (with the appropriate SQL statement).

This unit demonstrates both of these approaches. Figure 12.7 shows a high-level look at how the ADO.NET objects interact to make direct data access work.

Notes

Figure 12.7: Direct Data Access with ADO.NET



Before continuing, make sure you import the ADO.NET namespaces. In this unit, we assume you're using the SQL Server provider, in which case you need these two namespace imports:

```
using System.Data;
using System.Data.SqlClient;
```

12.5.1 Creating a Connection

Before you can retrieve or update data, you need to make a connection to the data source. Generally, connections are limited to some fixed number, and if you exceed that number (either because you run out of licenses or because your database server can't accommodate the user load), attempts to create new connections will fail. For that reason, you should try to hold a connection open for as short a time as possible. You should also write your database code inside a try/catch error-handling structure so you can respond if an error does occur, and make sure you close the connection even if you can't perform all your work.

When creating a Connection object, you need to specify a value for its ConnectionString property. This ConnectionString defines all the information the computer needs to find the data source, log in, and choose an initial database. Out of all the details in the examples in this unit, the ConnectionString is the one value you might have to tweak before it works for the database you want to use. Luckily, it's quite straightforward. Here's an example that uses a connection string to connect to SQL Server through the OLE DB provider:

```
OleDbConnection myConnection = new OleDbConnection();
myConnection.ConnectionString = "Provider=SQLOLEDB.1;Data Source=
localhost;" +
"Initial Catalog=Pubs;Integrated Security=SSPI";
```

For optimum performance, you should use the SqlConnection object from the SQL Server provider instead. The connection string for the SqlConnection object is quite similar and just omits the Provider setting:

```
SqlConnection myConnection = new SqlConnection();
myConnection.ConnectionString = "Data Source=localhost;" +
"Initial Catalog=Pubs;Integrated Security=SSPI";
```

Notes

If you're using SQL Server 2005 Express Edition, your connection string will include an instance name, as shown here:

```
SqlConnection myConnection = new SqlConnection();
myConnection.ConnectionString = @"Data Source=localhost\SQLEXPRESS;" +
"Initial Catalog=Pubs;Integrated Security=SSPI";
```

Notes When you add the instance name in C#, you must add two backslash characters, as in localhost\\SQLEXPRESS. This is because a single backslash is interpreted as a special character. However, if you define the connection string in a configuration file, as described in the next section, you need only one backslash, because you're no longer dealing with pure C# code.

The Connection String

The connection string is actually a series of distinct pieces of information separated by semicolons (;). Each separate piece of information is known as a connection string property.

The following list describes some of the most commonly used connection string properties, including the three properties used in the preceding example:

1. ***Data source:*** This indicates the name of the server where the data source is located. If the server is on the same computer that hosts the ASP.NET site, localhost is sufficient. The only exception is if you're using a named instance of SQL Server. For example, if you've installed SQL Server 2005 Express Edition, you'll need to use the data source localhost\SQLEXPRESS, because the instance name is SQLEXPRESS. You'll also see this written with a period, as .\SQLEXPRESS, which is equivalent.
2. ***Initial catalog:*** This is the name of the database that this connection will be accessing. It's only the "initial" database because you can change it later by using the Connection.ChangeDatabase() method.
3. ***Integrated security:*** This indicates you want to connect to SQL Server using the Windows user account that's running the web page code, provided you supply a value of SSPI (which stands for Security Support Provider Interface). Alternatively, you can supply a user ID and password that's defined in the database for SQL Server authentication, although this method is less secure and generally discouraged.
4. ***ConnectionTimeout:*** This determines how long your code will wait, in seconds, before generating an error if it cannot establish a database connection. Our example connection string doesn't set the ConnectionTimeout, so the default of 15 seconds is used. You can use 0 to specify no limit, but this is a bad idea. This means that, theoretically, the code could be held up indefinitely while it attempts to contact the server.

You can set some other, lesser-used options for a connection string. For more information, refer to the Visual Studio Help. Look under the appropriate Connection class (such as SqlConnection or OleDbConnection) because there are subtle differences in connection string properties for each type of Connection class.

Notes

Windows Authentication

The previous example uses integrated Windows authentication, which is the default security standard for new SQL Server installations. You can also use SQL Server authentication. In this case, you will explicitly place the user ID and password information in the connection string. However, SQL Server authentication is disabled by default in SQL Server 2000 and later versions, because it's not considered to be as secure.

Here's the lowdown on both types of authentication:

1. With SQL Server authentication, SQL Server maintains its own user account information in the database. It uses this information to determine whether you are allowed to access specific parts of a database.
2. With integrated Windows authentication, SQL Server automatically uses the Windows account information for the currently logged-in process. In the database, it stores information about what database privileges each user should have.

For Windows authentication to work, the currently logged-on Windows user must have the required authorization to access the SQL database. This isn't a problem while you test your websites, because Visual Studio launches your web applications using your user account. However, when you deploy your application to a web server running IIS, you might run into trouble. In this situation, all ASP.NET code is run by a more limited user account that might not have the rights to access the database. By default, that account is an automatically created account named ASPNET (for IIS 5.1), or the network service account (for later versions of IIS). You need to grant database access to this account, or your web pages will receive a security error whenever they try to connect to the database.

Notes If you're running IIS 5.1 (the version that's included with Windows XP), you need to give database access to the ASPNET user. If you're running IIS 6 (the version that's included with Windows Server 2003), you need to give access to the IIS_WPG group. If you're running IIS 7 (the version that's included with Windows Vista and Windows Server 2008), you need to give access to the IIS_USERS group.

User Instance Connections

Every database server stores a master list of all the databases that you've installed on it. This list includes the name of each database and the location of the files that hold the data. When you create a database (for example, by running a script or using a management tool), the information about that database is added to the master list. When you connect to the database, you specify the database name using the Initial Catalog value in the connection string.

Notes If you haven't made any changes to your database configuration, SQL Server will quietly tuck the files for newly created databases into a directory like c:\Program Files\Microsoft SQL Server\MSSQL.1\ MSSQL\Data. Each database has at least two files an .mdf file with the actual data and an .ldf file that stores the database log. Of course, database professionals have a variety of techniques and tricks for managing database storage, and can easily store databases in different locations, create multiple data files, and so on. The important detail to realize is that ordinarily your database files are stored by your database server, and they aren't a part of your web application directory.

Notes

Interestingly, SQL Server Express has a feature that lets you bypass the master list and connect directly to any database file, even if it's not in the master list of databases. This feature is called user instances. Oddly enough, this feature isn't available in the full edition of SQL Server 2005.

To use this feature, you need to set the User Instances value to True (in the connection string) and supply the file name of the database you want to connect to with the AttachDBFilename value. You don't supply an Initial Catalog value.

Example: Connection string that uses this approach:

```
myConnection.ConnectionString = @"Data Source=localhost\SQLEXPRESS;" +
@"AttachDBFilename=|DataDirectory|\Northwind.mdf;Integrated Security=True";
```

There's another trick here. The file name starts with |DataDirectory|. This automatically points to the App_Data folder inside your web application directory. This way, you don't need to supply a full file path, which might not remain valid when you move the web application to a web server. Instead, ADO.NET will always look in the App_Data directory for a file named Northwind.mdf.

User instances is a handy feature if you have a web server that hosts many different web applications that use databases and these databases are frequently being added and removed. However, because the database isn't in the master list, you won't see it in any administrative tools (although most administrative tools will still let you connect to it manually, by pointing out the right file location). But remember, this quirky but interesting feature is available in SQL Server Express only you won't find it in the full version of SQL Server 2005.

Storing the Connection String

Typically, all the database code in your application will use the same connection string. For that reason, it usually makes the most sense to store a connection string in a class member variable or, even better, a configuration file.

You can also create a Connection object and supply the connection string in one step by using a dedicated constructor:

```
SqlConnection myConnection = new SqlConnection(connectionString);
// myConnection.ConnectionString is now set to connectionString.
```

You don't need to hard-code a connection string. The <connectionStrings> section of the web.config file is a handy place to store your connection strings. Here's an example:

```
<configuration>
<connectionStrings>
<add name="Pubs" connectionString=
"Data Source=localhost;Initial Catalog=Pubs;Integrated Security=SSPI"/>
</connectionStrings>
...
</configuration>
```

Notes

You can then retrieve your connection string by name. First, import the System.Web.Configuration namespace. Then, you can use code like this:

```
string connectionString =
WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
```

This approach helps to ensure all your web pages are using the same connection string. It also makes it easy for you to change the connection string for an application, without needing to edit the code in multiple pages. The examples in this unit all store their connection strings in the web.config file in this way.

Making the Connection

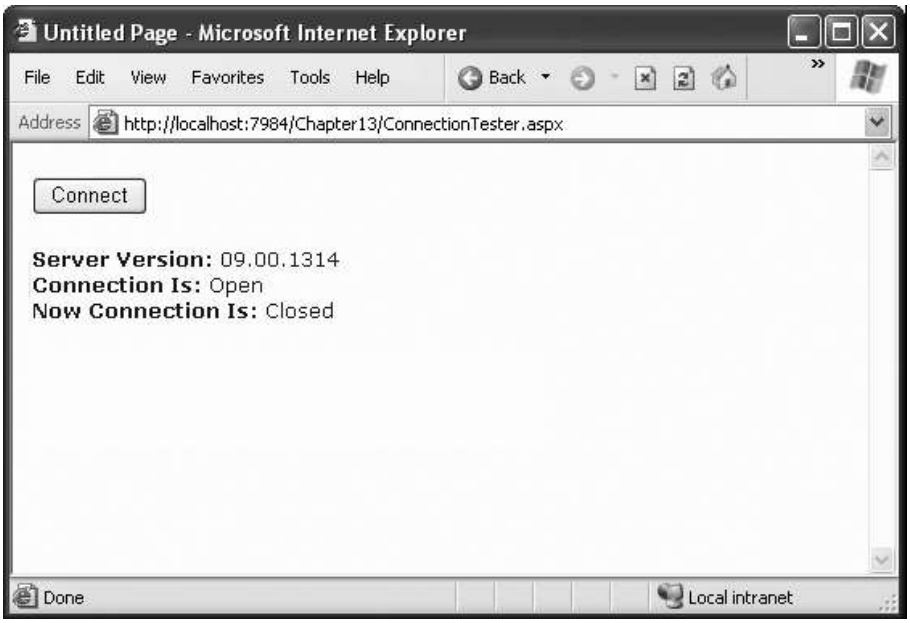
Once you've created your connection (as described in the previous section), you're ready to use it.

Before you can perform any database operations, you need to explicitly open your connection:

```
myConnection.Open();
```

To verify that you have successfully connected to the database, you can try displaying some basic connection information. The following example writes some basic information to a Label control named lblInfo (Figure 12.8).

Figure 12.8: Testing your Connection



Here's the code with basic error handling:

```
// Define the ADO.NET Connection object.
string connectionString =
WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
SqlConnection myConnection = new SqlConnection(connectionString);
```

Notes

```
try
{
// Try to open the connection.
myConnection.Open();
lblInfo.Text = "<b>Server Version:</b> " + myConnection.ServerVersion;
lblInfo.Text += "<br /><b>Connection Is:</b> " +
myConnection.State.ToString();
}
catch (Exception err)
{
// Handle an error by displaying the information.
lblInfo.Text = "Error reading the database. ";
lblInfo.Text += err.Message;
}
finally
{
// Either way, make sure the connection is properly closed.
// (Even if the connection wasn't opened successfully,
// calling Close() won't cause an error.)
myConnection.Close();
lblInfo.Text += "<br /><b>Now Connection Is:</b> ";
lblInfo.Text += myConnection.State.ToString();
}
```

Once you use the Open() method, you have a live connection to your database. One of the most fundamental principles of data access code is that you should reduce the amount of time you hold a connection open as much as possible. Imagine that as soon as you open the connection, you have a live, ticking time bomb. You need to get in, retrieve your data, and throw the connection away as quickly as possible in order to ensure your site runs efficiently.

Closing a connection is just as easy, as shown here:

```
myConnection.Close();
```

Another approach is to use the using statement. The using statement declares that you are using a disposable object for a short period of time. As soon as you finish using that object and the using block ends, the common language runtime will release it immediately by calling the Dispose() method. Here's the basic structure of the using block:

```
using (object)
{
...
}
```

Notes

It just so happens that calling the Dispose() method of a connection object is equivalent to calling Close(). That means you can shorten your database code with the help of a using block. The best part is that you don't need to write a finally block—the using statement releases the object you're using even if you exit the block as the result of an unhandled exception.

Here's how you could rewrite the earlier example with a using block:

```
SqlConnection myConnection = new SqlConnection(connectionString);

try
{
using (myConnection)
{
// Try to open the connection.
myConnection.Open();
lblInfo.Text = "<b>Server Version:</b> " + myConnection.ServerVersion;
lblInfo.Text += "<br /><b>Connection Is:</b> " +
myConnection.State.ToString();
}
catch (Exception err)
{
// Handle an error by displaying the information.
lblInfo.Text = "Error reading the database. ";
lblInfo.Text += err.Message;
}
lblInfo.Text += "<br /><b>Now Connection Is:</b> ";
lblInfo.Text += myConnection.State.ToString();
```

There's one difference in the way this code is implemented as compared to the previous example. The error-handling code wraps the using block. As a result, if an error occurs the database connection is closed first, and then the exception-handling code is triggered. In the first example, the error-handling code responded first, and then the finally block closed the connection afterward. Obviously, this rewrite is a bit better, as it's always good to close database connections as soon as possible.

12.5.2 The Select Command

The Connection object provides a few basic properties that supply information about the connection, but that's about all. To actually retrieve data, you need a few more ingredients:

1. A SQL statement that selects the information you want
2. A Command object that executes the SQL statement
3. A DataReader or DataSet object to access the retrieved records

Command objects represent SQL statements. To use a Command, you define it, specify the SQL statement you want to use, specify an available connection, and execute the command.

Notes

You can use one of the earlier SQL statements, as shown here:

```
SqlCommand myCommand = new SqlCommand();
myCommand.Connection = myConnection;
myCommand.CommandText = "SELECT * FROM Authors ORDER BY au_lname ";
```

Or you can use the constructor as a shortcut:

```
SqlCommand myCommand = new SqlCommand(
"SELECT * FROM Authors ORDER BY au_lname ", myConnection);
```

Notes It's also a good idea to dispose of the Command object when you're finished, although it isn't as critical as closing the Connection object.

12.5.3 The DataReader

Once you've defined your command, you need to decide how you want to use it. The simplest approach is to use a DataReader, which allows you to quickly retrieve all your results. The DataReader uses a live connection and should be used quickly and then closed. The DataReader is also extremely simple. It supports fast-forward-only read-only access to your results, which is generally all you need when retrieving information. Because of the DataReader's optimized nature, it provides better performance than the DataSet. It should always be your first choice for direct data access.

Before you can use a DataReader, make sure you've opened the connection:

```
myConnection.Open();
```

To create a DataReader, you use the ExecuteReader() method of the command object, as shown here:

```
// You don't need the new keyword, as the Command will create the DataReader.
SqlDataReader myReader;
myReader = myCommand.ExecuteReader();
```

These two lines of code define a variable for a DataReader and then create it by executing the command. Once you have the reader, you retrieve a single row at a time using the Read() method:

```
myReader.Read(); // The first row in the result set is now available.
```

You can then access the values in the current row using the corresponding field names.

The following example adds an item to a list box with the first name and last name for the current row:

```
lstNames.Items.Add(myReader["au_lname"] + ", " + myReader["au_fname"]);
```

To move to the next row, use the Read() method again. If this method returns True, a row of information has been successfully retrieved. If it returns False, you've attempted to read past the end of your result set. There is no way to move backward to a previous row.

Notes

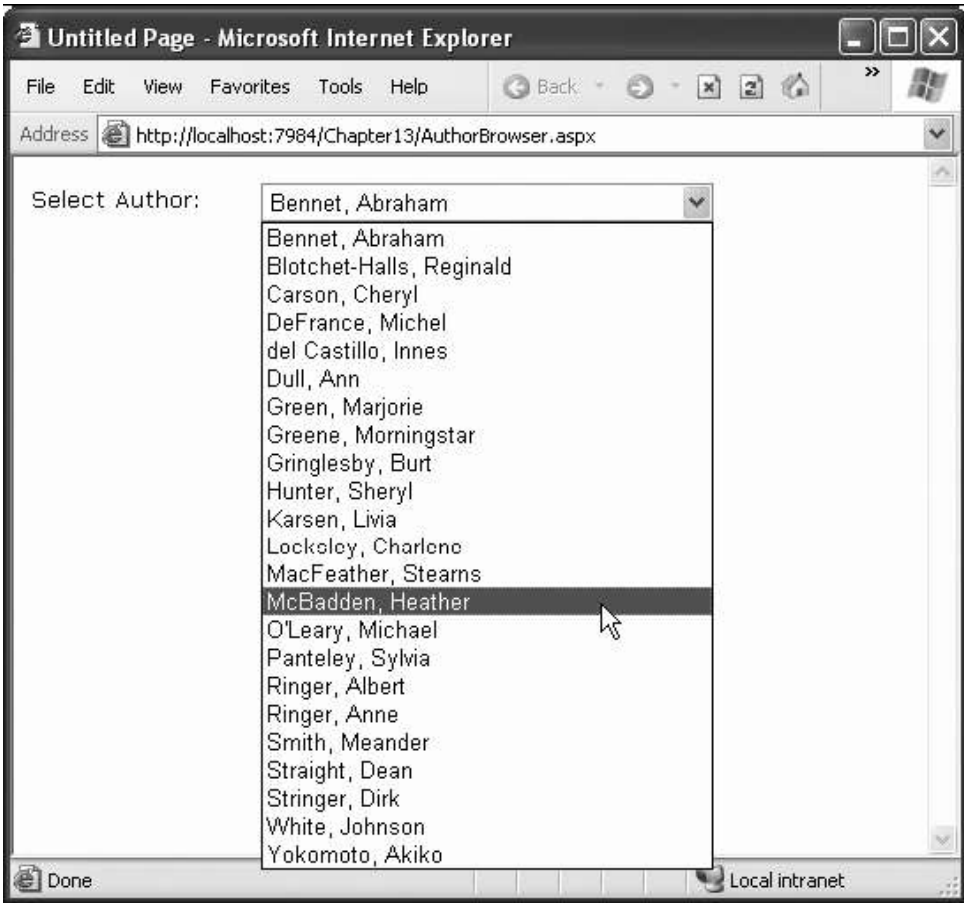
As soon as you've finished reading all the results you need, close the DataReader and Connection:

```
myReader.Close();
myConnection.Close();
```

Putting it all Together

The next example demonstrates how you can use all the ADO.NET ingredients together to create a simple application that retrieves information from the Authors table. You can select an author record by last name using a drop-down list box, as shown in Figure 12.9.

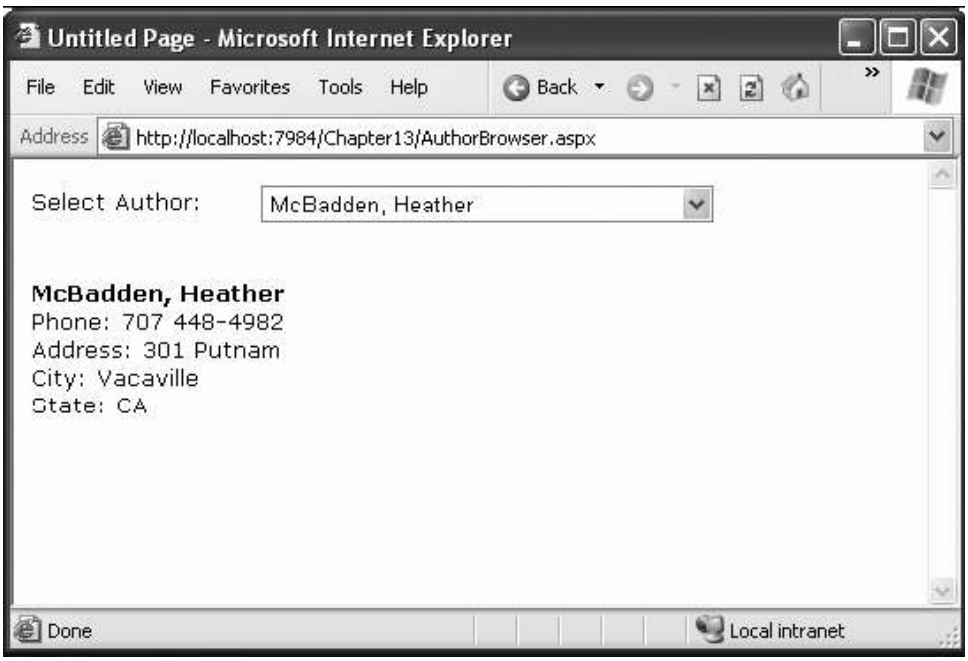
Figure 12.9: Selecting an Author



The full record is then retrieved and displayed in a simple label, as shown in Figure 12.10.

Notes

Figure 12.10: Author Information



Filling the List Box

To start, the connection string is defined as a private variable for the page class and retrieved from the connection string:

```
private string connectionString =
WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
```

The list box is filled when the Page.Load event occurs. Because the list box is set to persist its view state information, this information needs to be retrieved only once the first time the page is displayed. It will be ignored on all postbacks.

Here's the code that fills the list from the database:

```
protected void Page_Load(Object sender, EventArgs e)
{
if (!this.IsPostBack)
{
FillAuthorList();
}
}
private void FillAuthorList()
{
lstAuthor.Items.Clear();
// Define the Select statement.
// Three pieces of information are needed: the unique id
```

Notes

```
// and the first and last name.
string selectSQL = "SELECT au_lname, au_fname, au_id FROM Authors";
// Define the ADO.NET objects.
SqlConnection con = new SqlConnection(connectionString);
SqlCommand cmd = new SqlCommand(selectSQL, con);
SqlDataReader reader;
// Try to open database and read information.
try
{
con.Open();
reader = cmd.ExecuteReader();
// For each item, add the author name to the displayed
// list box text, and store the unique ID in the Value property.
while (reader.Read())
{
ListItem newItem = new ListItem();
newItem.Text = reader["au_lname"] + ", " + reader["au_fname"];
newItem.Value = reader["au_id"].ToString();
lstAuthor.Items.Add(newItem);
}
reader.Close();
}
catch (Exception err)
{
lblResults.Text = "Error reading list of names. ";
lblResults.Text += err.Message;
}
finally
{
con.Close();
}
}
```

This example looks more sophisticated than the previous bite-sized snippets in this unit, but it really doesn't introduce anything new. It uses the standard Connection, Command, and DataReader objects. The Connection is opened inside an error-handling block so your page can handle any unexpected errors and provide information. A finally block makes sure the connection is properly closed, even if an error occurs.

The actual code for reading the data uses a loop. With each pass, the Read() method is called to get another row of information. When the reader has read all the available information, this method will return false, the loop condition will evaluate to false, and the loop will end gracefully.

The unique ID (the value in the au_id field) is stored in the Value property of the list box for reference later. This is a crucial ingredient that is needed to allow the corresponding record to be queried again. If you tried to build a query using the author's name, you would need to worry about authors with the same name. You would also have the additional headache of invalid characters (such as the apostrophe in O'Leary) that would invalidate your SQL statement.

Notes

Retrieving the Record

The record is retrieved as soon as the user changes the selection in the list box. To make this possible, the AutoPostBack property of the list box is set to true so that its change events are detected automatically.

```
protected void lstAuthor_SelectedIndexChanged(Object sender, EventArgs e)
{
// Create a Select statement that searches for a record
// matching the specific author ID from the Value property.
string selectSQL;
selectSQL = "SELECT * FROM Authors ";
selectSQL += "WHERE au_id='" + lstAuthor.SelectedItem.Value + "'";

// Define the ADO.NET objects.
SqlConnection con = new SqlConnection(connectionString);
SqlCommand cmd = new SqlCommand(selectSQL, con);
SqlDataReader reader;
// Try to open database and read information.
try
{
con.Open();
reader = cmd.ExecuteReader();
reader.Read();
// Build a string with the record information,
// and display that in a label.
StringBuilder sb = new StringBuilder();
sb.Append("<b>");
sb.Append(reader["au_lname"]);
sb.Append(", ");
sb.Append(reader["au_fname"]);
sb.Append("</b><br />");
sb.Append("Phone: ");
sb.Append(reader["phone"]);
sb.Append("<br />");
sb.Append("Address: ");
sb.Append(reader["address"]);
sb.Append("<br />");
sb.Append("City: ");
sb.Append(reader["city"]);
sb.Append("<br />");
sb.Append("State: ");
sb.Append(reader["state"]);
sb.Append("<br />");
lblResults.Text = sb.ToString();
reader.Close();
```

Notes

```
}
catch (Exception err)
{
lblResults.Text = "Error getting author. ";
lblResults.Text += err.Message;
}
finally
{
con.Close();
}
}
```

The process is similar to the procedure used to retrieve the last names. There are only a couple of differences:

1. The code dynamically creates a SQL statement based on the selected item in the dropdown list box. It uses the Value property of the selected item, which stores the unique identifier. This is a common (and useful) technique.
2. Only one record is read. The code assumes that only one author has the matching au_id, which is reasonable since this field is unique.

Task Discuss how will you update a record in table using ASP programming.

12.6 Single-value Data Binding

Single-value data binding is really just a different approach to dynamic text. To use it, you add special data binding expressions into your .aspx files. These expressions have the following format:

```
<%# expression_goes_here %>
```

This may look like a script block, but it isn't. If you try to write any code inside this tag, you will receive an error. The only thing you can add is a valid data binding expression. For example, if you have a public or protected variable named Country in your page, you couldwrite the following:

```
<%# Country %>
```

When you call the DataBind() method for the page, this text will be replaced with the value for Country (for example, Spain). Similarly, you could use a property or a built-in ASP.NET object as follows:

```
<%# Request.Browser.Browser %>
```

This would substitute a string with the current browser name (for example, IE). In fact, you can even call a function defined on your page, or execute a simple expression, provided it returns a result that can be converted to text and displayed on the page. Thus, the following data binding expressions are all valid:

```
<%# GetUserName(ID) %>
```

Notes

```
<%# 1 + (2 * 20) %>
<%# "John " + "Smith" %>
```

Remember, you place these data binding expressions in the markup portion of your .aspx file, not your code-behind file.

12.6.1 A Simple Data Binding Example

This section shows a simple example of single-value data binding. The example has been stripped to the bare minimum amount of detail needed to illustrate the concept.

You start with a variable defined in your Page class, which is called TransactionCount:

```
public partial class SimpleDataBinding : System.Web.UI.Page
{
protected int TransactionCount;
// (Additional code omitted.)
}
```

Note that this variable must be designated as public, protected, or internal, but not private. If you make the variable private, ASP.NET will not be able to access it when it's evaluating the data binding expression.

Now, assume that this value is set in the Page.Load event handler using some database lookup code. For testing purposes, the example skips this step and hard-codes a value:

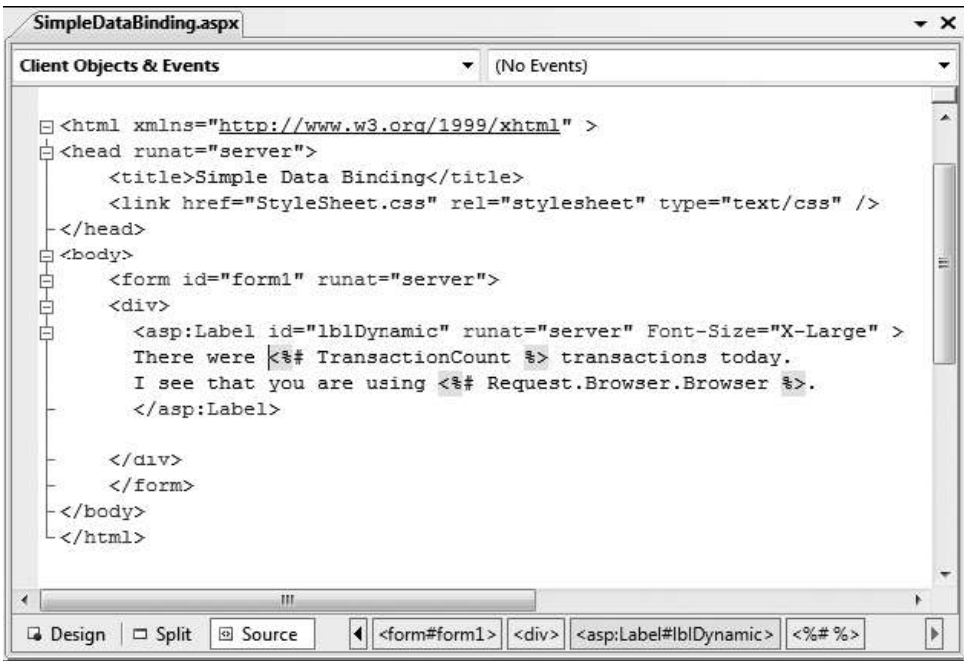
```
protected void Page_Load(object sender, EventArgs e)
{
// (You could use database code here
// to look up a value for TransactionCount.)
TransactionCount = 10;

// Now convert all the data binding expressions on the page.
this.DataBind();
}
```

Two actions actually take place in this event handler: the TransactionCount variable is set to 10, and all the data binding expressions on the page are bound. Currently, no data binding expressions exist, so this method has no effect. Notice that this example uses the this keyword to refer to the current page. You could just write DataBind() without the this keyword, because the default object is the current Page object. However, using the this keyword makes it a bit clearer what object is being used.

To make this data binding accomplish something, you need to add a data binding expression. Usually, it's easiest to add this value directly to the markup in the .aspx file. To do so, click the Source button at the bottom of the web page designer window. Figure 12.11 shows an example with a Label control.

Notes

Figure 12.11: Source View in the Web Page Designer

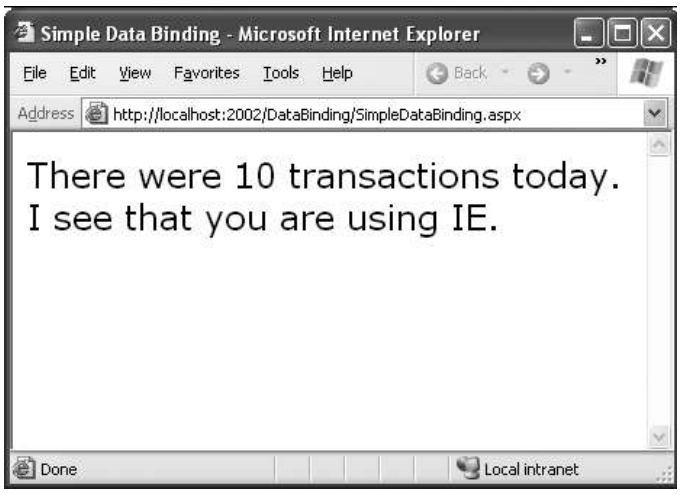


To add your expression, find the tag for the Label control. Modify the text inside the label as shown here:

```
<asp:Label id="lblDynamic" runat="server" Font-Size="X-Large">
There were <%# TransactionCount %> transactions today.
I see that you are using <%# Request.Browser.Browser %>.
</asp:Label>
```

This example uses two separate data binding expressions, which are inserted along with the normal static text. The first data binding expression references the TransactionCount variable, and the second uses the built-in Request object to determine some information about the user's browser. When you run this page, the output looks like Figure 12.12.

Figure 12.12: The Result of Data Binding

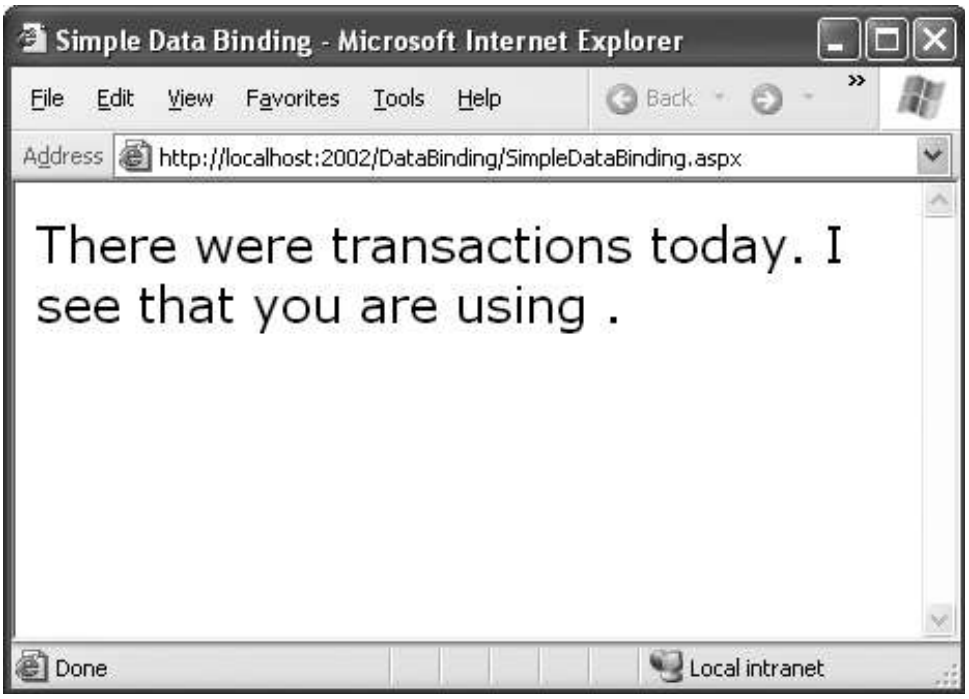


Notes

The data binding expressions have been automatically replaced with the appropriate values. If the page is posted back, you could use additional code to modify TransactionCount, and as long as you call the DataBind() method, that information will be popped into the page in the data binding expression you've defined.

If, however, you forget to call the DataBind() method, the data binding expressions will be ignored, and the user will see a somewhat confusing window that looks like Figure 12.13.

Figure 12.13: The Non-data-bound Page



Notes When using single-value data binding, you need to consider when you should call the DataBind() method. For example, if you made the mistake of calling it before you set the TransactionCount variable, the corresponding expression would just be converted to 0. Remember, data binding is a one-way street. This means changing the TransactionCount variable after you've used the DataBind() method won't produce any visible effect. Unless you call the DataBind() method again, the displayed value won't be updated.

12.6.2 Simple Data Binding with Properties

The previous example uses a data binding expression to set static text information inside a label tag. However, you can also use single-value data binding to set other types of information on your page, including control properties. To do this, you simply have to know where to put the data binding expression in the web page markup.

Notes

Example: Consider the following page, which defines a variable named URL and uses it to point to a picture in the application directory:

```
public partial class DataBindingUrl : System.Web.UI.Page
{
public string URL;
protected void Page_Load(Object sender, EventArgs e)
{
URL = "Images/picture.jpg";
this.DataBind();
}
}
```

You can now use this URL to create a label, as shown here:

```
<asp:Label id="lblDynamic" runat="server"><%# URL %></asp:Label>
```

You can also use it for a check box caption:

```
<asp:CheckBox id="chkDynamic" Text="<%# URL %>" runat="server" />
```

or you can use it for a target for a hyperlink:

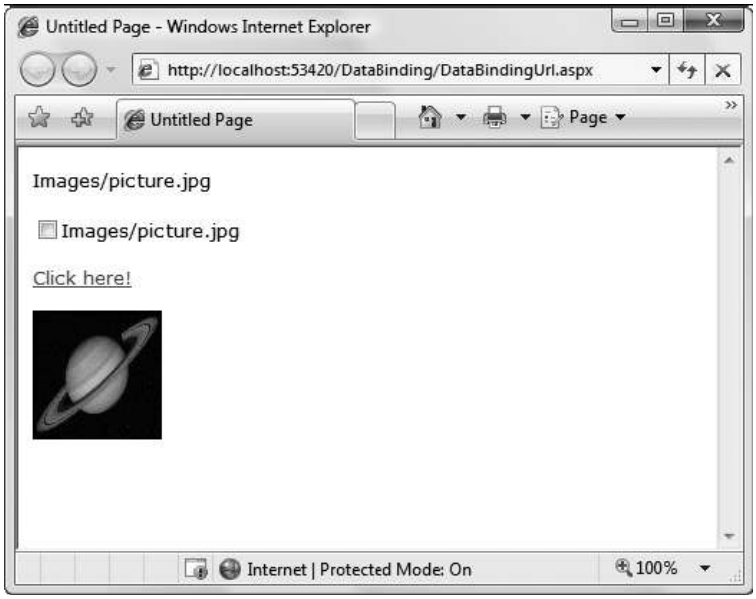
```
<asp:Hyperlink id="lnkDynamic" Text="Click here!" NavigateUrl="<%# URL %>"
runat="server" />
```

You can even use it for a picture:

```
<asp:Image id="imgDynamic" ImageUrl="<%# URL %>" runat="server" />
```

The only trick is that you need to edit these control tags manually. Figure 12.14 shows what a page that uses all these elements would look like.

Figure 12.14: Multiple Ways to Bind the same Data



Notes

12.6.3 Problems with Single-value Data Binding

Before you start using single-value data binding techniques in every aspect of your ASP.NET programs, you should consider some of the serious drawbacks this approach can present:

1. ***Putting code into a page's user interface***: One of ASP.NET's great advantages is that it allows developers to separate the user interface code (the HTML and control tags in the .aspx file) from the actual code used for data access and all other tasks (in the code-behind file). However, overenthusiastic use of single-value data binding can encourage you to disregard that distinction and start coding function calls and even operations into your page. If not carefully managed, this can lead to complete disorder.
2. ***Fragmenting code***: When using data binding expressions, it may not be obvious where the functionality resides for different operations. This is particularly a problem if you blend both approaches—for example, if you use data binding to fill a control and also modify that control directly in code. Even worse, the data binding code may have certain dependencies that aren't immediately obvious. If the page code changes, or a variable or function is removed or renamed, the corresponding data binding expression could stop providing valid information without any explanation or even an obvious error. All of these details make it more difficult to maintain your code, and make it more difficult for multiple developers to work together on the same project.

Of course, some developers love the flexibility of single-value data binding and use it to great effect, making the rest of their code more economical and streamlined. It's up to you to be aware of (and avoid) the potential drawbacks.

Notes In one case, single-value data binding is quite useful—when building templates. Templates declare a block of markup that's reused for each record in a table. However, they work only with certain rich data controls, such as the GridView.

12.7 Repeated-value Data Binding

Although using simple data binding is optional, repeated-value binding is so useful that almost every ASP.NET application will want to use it somewhere.

Repeated-value data binding works with the ASP.NET list controls (and the rich data controls described in the next unit). To use repeated-value binding, you link one of these controls to a data source (such as a field in a data table). When you call DataBind(), the control automatically creates a full list using all the corresponding values. This saves you from writing code that loops through the array or data table and manually adds elements to a control.

Repeated-value binding can also simplify your life by supporting advanced formatting and template options that automatically configure how the data should look when it's placed in the control.

To create a data expression for list binding, you need to use a list control that explicitly supports data binding. Luckily, ASP.NET provides a number of list controls, many of which you've probably already used in other applications or examples:

1. ***ListBox, DropDownList, CheckBoxList, and RadioButtonList:*** These web controls provide a list for a single field of information.
2. ***HtmlSelect:*** This server-side HTML control represents the HTML <select> element and works essentially the same way as the ListBox web control. Generally, you'll use this control only for backward compatibility.

Notes

3. ***GridView,DetailsView, FormView, and ListView:*** These rich web controls allow you to provide repeating lists or grids that can display more than one field of information at a time.

Example: If you bind one of these controls to a full-fledged table in a DataSet, you can display the values from multiple fields. These controls offer the most powerful and flexible options for data binding.

With repeated-value data binding, you can write a data binding expression in your .aspx file, or you can apply the data binding by setting control properties. In the case of the simpler list controls, you'll usually just set properties. Of course, you can set properties in many ways, such as by using code in a code-behind file or by modifying the control tag in the .aspx file, possibly with the help of Visual Studio's Properties window. The approach you take doesn't matter. The important detail is that you don't use any <%# expression %> data binding expressions.

To continue any further with data binding, it will help to divide the subject into a few basic categories. You'll start by looking at data binding with the list controls.

12.7.1 Data Binding with Simple List Controls

In some ways, data binding to a list control is the simplest kind of data binding. You need to follow only three steps:

1. ***Create and fill some kind of data object.*** You have numerous options, including an array, the basic ArrayList and Hashtable collections, the strongly typed List and Dictionary collections, and the ADO.NET DataTable and DataSet objects. Essentially, you can use any type of collection that supports the IEnumerable interface, although you'll discover each class has specific advantages and disadvantages.
2. ***Link the object to the appropriate control.*** To do this, you need to set only a couple of properties, including DataSource. If you're binding to a full DataSet, you'll also need to set the DataMember property to identify the appropriate table you want to use.
3. ***Activate the binding.*** As with single-value binding, you activate data binding by using the DataBind() method, either for the specific control or for all contained controls at once by using the DataBind() method for the current page.

This process is the same whether you're using the ListBox, the DropDownList, the CheckBoxList, the RadioButtonList, or even the HtmlSelect control. All these controls provide the same properties and work the same way. The only difference is in the way they appear on the final web page.

12.7.2 A Simple List Binding Example

To try this type of data binding, add a ListBox control to a new web page. Next, import the System.Collections namespace in your code. Finally, use the Page.Load event handler to create an ArrayList collection to use as a data source as follows:

```
ArrayList fruit = new ArrayList();
fruit.Add("Kiwi");
fruit.Add("Pear");
fruit.Add("Mango");
fruit.Add("Blueberry");
```

Notes

```
fruit.Add("Apricot");
fruit.Add("Banana");
fruit.Add("Peach");
fruit.Add("Plum");
```

Now, you can link this collection to the ListBox control:

```
lstItems.DataSource = fruit;
```

Because an ArrayList is a straightforward, unstructured type of object, this is all the information you need to set. If you were using a DataTable (which has more than one field) or a DataSet (which has more than one DataTable), you would have to specify additional information.

To activate the binding, use the DataBind() method:

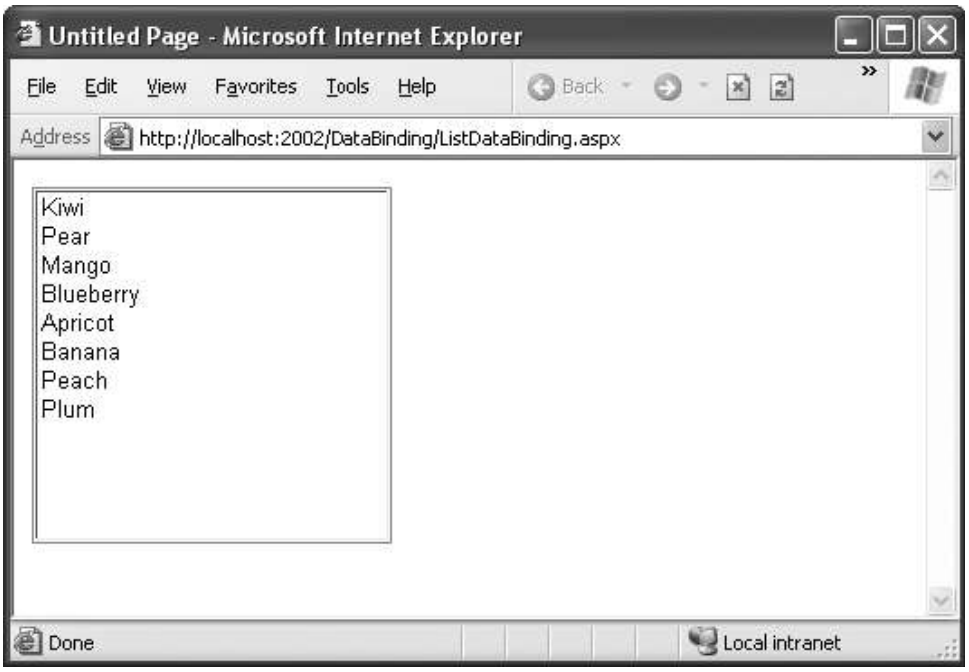
```
this.DataBind();
```

You could also use lstItems.DataBind() to bind just the ListBox control. Figure 12.15 shows the resulting web page.

This technique can save quite a few lines of code. This example doesn't offer a lot of savings because the collection is created just before it's displayed. In a more realistic application, however, you might be using a function that returns a ready-made collection to you:

```
ArrayList fruit;
fruit = GetFruitsInSeason("Summer");
```

Figure 12.15: A Data-bound List



Notes

In this case, it's extremely simple to add the extra two lines needed to bind and display the collection in the window:

```
lstItems.DataSource = fruit;
this.DataBind();
```

or you could even change it to the following, even more compact, code:

```
lstItems.DataSource = GetFruitsInSeason("Summer");
this.DataBind();
```

On the other hand, consider the extra trouble you would have to go through if you didn't use data binding. This type of savings compounds rapidly, especially when you start combining data binding with multiple controls, advanced objects such as DataSets, or advanced controls that apply formatting through templates.

12.7.3 Strongly Typed Collections

You can use data binding with the Hashtable and ArrayList, two of the more useful collection classes in the System.Collections namespace.

System.Collections.Generic. These collections are ideal in cases where you want your collection to hold just a single type of object (for example, just strings). When you use the generic collections, you choose the item type you want to use, and the collection object is "locked in" to your choice (which is similar to how an array works). This means if you try to add another type of object that doesn't belong in the collection, you'll get a compile-time error. Similarly, when you pull an item out of the collection, you don't need to write casting code to convert it to the right type, because the compiler already knows what type of objects you're using. This behavior is safer and more convenient, and it's what you'll want most of the time. To use a generic collection, you must import the right namespace:

```
using System.Collections.Generic
```

The generic version of the ArrayList class is named List. Here's how you create a List collection object that can only store strings:

```
List<string> fruit = new List<string>();
fruit.Add("Kiwi");
fruit.Add("Pear");
```

The only real difference is that you need to specify the type of data you want to use when you declare the List object.

12.7.4 Multiple Binding

You can bind the same data list object to multiple different controls. Consider the following example, which compares all the types of list controls at your disposal by loading them with the same information:

```
protected void Page_Load(Object sender, EventArgs e)
{
// Create and fill the collection.
List<string> fruit = new List<string>();
fruit.Add("Kiwi");
fruit.Add("Pear");
```

Notes

```
fruit.Add("Mango");
fruit.Add("Blueberry");
fruit.Add("Apricot");
fruit.Add("Banana");
fruit.Add("Peach");
fruit.Add("Plum");

// Define the binding for the list controls.
MyListBox.DataSource = fruit;
MyDropDownListBox.DataSource = fruit;
MyHtmlSelect.DataSource = fruit;
MyCheckBoxList.DataSource = fruit;
MyRadioButtonList.DataSource = fruit;

// Activate the binding.
this.DataBind();
}
```

Figure 12.16 shows the rendered page.

Figure 12.16: Multiple Bound Lists

Notes

This is another area where ASP.NET data binding may differ from what you have experienced in a desktop application. In traditional data binding, all the different controls are sometimes treated like "views" on the same data source, and you can work with only one record from the data source at a time. In this type of data binding, when you select Pear in one list control, the other list controls automatically refresh so that they too have Pear selected (or the corresponding information from the same row). This isn't how ASP.NET uses data binding. If you want this sort of effect, you need to write custom code to pull it off.

Case Study

Building the IT Infrastructure

Alfred is a do-it-yourself entrepreneur who built up his fortune in trading. He traded in anything and everything, and kept close control of every activity. That was how he had grown rich enough to indulge in his one dream — to build a college in his hometown. A college that would be at par to the ones in the better cities, the ones in which he could not study himself.

Work started a year back and the buildings were coming along well. He himself did not use computers much and became hooked to the Internet and e-mail only recently. He was determined to provide a PC with Internet connectivity to every students and faculty member. He was currently engrossed in plans for the 100 - seater computer lab.

What was confusing him was the choice of Internet connectivity. He had about a dozen quotes in front of him. Recommendations ranged from 64 kbps ISDN all the way to 1 Gbps leased line to Guwahati, which was almost 200 km away. Prices ranged from slightly under a lakh all the way up to ₹ 25 lakh and beyond. He did not understand most of the equipment quoted — firewall, proxy server, cache appliance. Nor was he sure what the hidden costs were. Although it went against his very nature, he would have to identify a trustworthy consultant who would help him make sense of the whole thing.

Questions

1. In the context of the given case, what managerial issues need to be addressed by Alfred? Why is it important for managers to be tech savvy?
2. What is the importance of a 'systems consultant' to an organization? What skills should he/she possess?

12.8 Summary

- The basic principle of data binding is this: you tell a control where to find your data and how you want it displayed, and the control handles the rest of the details.
- Data binding in ASP.NET is superficially similar to data binding in the world of desktop or client/server applications, but in truth, it's fundamentally different. In those environments, data binding involves creating a direct connection between a data source and a control in an application window.
- If the user changes a value in the on-screen control, the data in the linked database is modified automatically.
- Similarly, if the database changes while the user is working with it (for example, another user commits a change), the display can be refreshed automatically.

Notes

12.9 Keywords

ADO.NET (ActiveX Data Objects for .NET): The set of .NET classes and data providers used to manipulate databases, such as Microsoft SQL Server 2000. ADO.NET was formerly known as ADO+. ADO.NET can be used by any .NET language.

Data Binding: The process or method for configuring controls on a form or Web page to fetch data from or write data to a data source such as a database, XML file, and so on.

Data Source Control: An object that can be added to an ASP.NET Web page that encapsulates the logic required to connect to a data source, such as a database or XML file, and that can execute queries or other data-access commands.

DCOM (Distributed Component Object Model): An extension of the Microsoft Component Object Model (COM) that allows COM components to communicate across network boundaries.

Dynamic Data: In ASP.NET, a framework that facilitates the creation of data-driven Web applications. Dynamic Data uses customizable page and field templates, scaffolding, user-definable metadata, and convention-based naming to create UI that displays data, lets users navigate relationships between tables, and lets users edit and create data (CRUD operations).

12.10 Self Assessment

Fill in the blanks:

1. ODBC stands for .....
2. RDO stands for .....
3. XML stands for .....
4. ADO.NET has two central components data providers and .....
5. The .NET data provider for SQL Server is located in the ..... namespace.
6. ..... is a scaled-down version of SQL Server 2005 that's free to distribute.
7. ADO.NET has three distinct types of classes commonly referred as Disconnected, ... and Data Providers.

State whether the following statements are True or False:

8. A data provider does not contain Connection, Command, DataAdapter, and DataReader objects.
9. The Connection object creates a link to the data source.
10. The DataReader object is a simple forward-only and read-only cursor.
11. The SqlDataAdapter provides a single-day data transfer mechanism.
12. The connection string is actually a series of distinct pieces of information separated by semicolons (;)

12.11 Review Questions

1. What type of control do you need to retrieve data from the database?
2. What is the name of the process for allowing a control, such as a GridView, to extract data from the retrieved tables and format it properly?

Notes

3. What is a connection string?
4. How do you attach a data source to a GridView?
5. If your table has many rows, what should you do in the GridView to make it easier to read?
6. How can you enable users to change the contents of the database from your GridView?
7. How can you take an action based on the data in a row, as the table is loaded?
8. If you're using a using a LinqDataSource control connected to a GridView, and you want users to be able to edit the data in the database, how do you need to configure the data that the LinqDataSource retrieves?
9. What are templates, in terms of Dynamic Data?
10. What do you mean by debugging connections to SQL server?
11. Describe connection pooling.
12. Explain briefly connecting Microsoft access with OLEDB connection.
13. Describe connection with MYSQL.

Answers: Self Assessment

1.	Open DataBase Connectivity	2.	Remote Data Objects
3.	Extensible Markup Language	4.	datasets
5.	System.Data.SqlClient	6.	SQL Server Express
7.	Shared	8.	False
9.	True	10.	True
11.	False	12.	True

12.12 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Notes

Unit 13: Website Security

CONTENTS

Objectives

Introduction

13.1 ASP.NET Security Model

13.2 Authentication and Authorization

13.2.1 Applying Authentication Measures

13.2.2 The <authentication> Node

13.3 Forms Authentication

13.3.1 Authenticating against Values contained in the web.config File

13.3.2 Authenticating against Values in a Database

13.3.3 Using the Login Control with Forms Authentication

13.3.4 Forms Authentication Class

13.4 Windows Authentication

13.4.1 Creating Users

13.4.2 Authenticating and Authorizing a User

13.4.3 Authenticating and Authorizing a Group

13.4.4 Authenticating and Authorizing an HTTP Transmission Method

13.4.5 Integrated Windows Authentication

13.4.6 Basic Authentication

13.4.7 Digest Authentication

13.5 Summary

13.6 Keywords

13.7 Self Assessment

13.8 Review Questions

13.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe ADO.NET security model
- Explain form authentication
- Know windows authentication

Notes

Introduction

Not every page that you build with ASP.NET is meant to be open and accessible to everyone on the Internet. Sometimes, you want to build pages or sections of an application that are accessible to only a select group of your choosing. For this reason, you need the security measures explained in this unit. They can help protect the data behind your applications and the applications themselves from fraudulent use.

Security is a very wide-reaching term. During every step of the application-building process, you must, without a doubt, be aware of how mischievous end users might attempt to bypass your lockout measures. You must take steps to ensure that no one can take over the application or gain access to its resources. Whether it involves working with basic server controls or accessing databases, you should be thinking through the level of security you want to employ to protect yourself.

How security is applied to your applications is truly a measured process. For instance, a single ASP.NET page on the Internet, open to public access, has different security requirements than does an ASP.NET application that is available to only selected individuals because it deals with confidential information such as credit card numbers or medical information.

The first step is to apply the appropriate level of security for the task at hand. Because you can take so many different actions to protect your applications and the resources, you have to decide for yourself which of these measures to employ. This unit takes a look at some of the possibilities for protecting your applications.

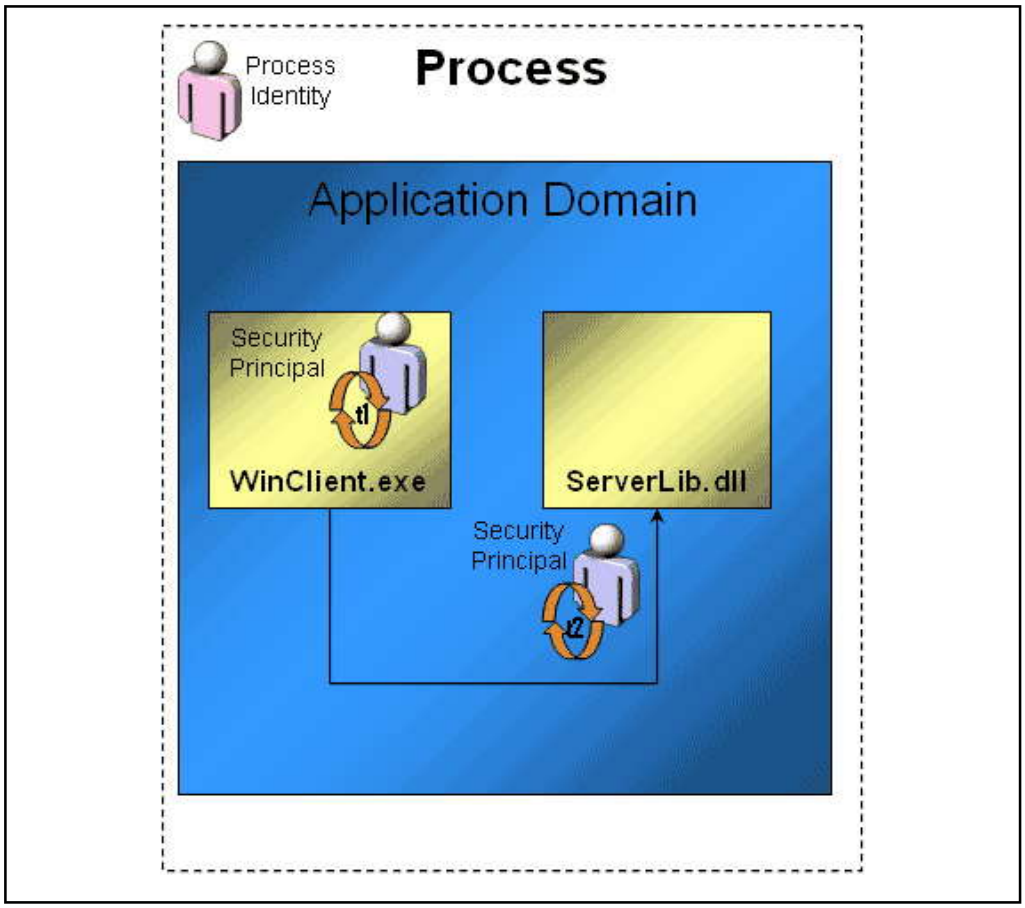
13.1 ASP.NET Security Model

Role-based security is built on the premise that users are authenticated, which is the process of identifying the user. Once identified, the user can be authorized or, assigned roles and permissions. Credentials like a username and password are usually provided to authenticate users, and this information is used to create a security principal representing this user's identity at runtime. The .NET Framework object model includes built-in support to work with Windows, custom and even Microsoft Passport credentials.

Let's assume that credentials have been collected from the user, and that a security principal has been created. To understand how this security principal is used by the runtime it is important to consider the relationship between the running process, the application domain, and the assemblies loaded within that application domain. In addition, we must consider how one or more logical threads of execution are handled. In figure shows a single process with a Windows executable (WinClient.exe) hosted within an application domain. The main thread (t1) may start another thread (t2) to invoke a method in another DLL assembly (ServerLib.dll).

By default the process runs under the logged in user's Windows identity, and this governs what resources can be accessed by any thread of execution within that process. If a method invoked by the execution context of t1 or t2 attempts to write to the file system, the process identity governs its privileges. Yet, each thread of execution can also be assigned an identity which governs how role-based security checks are evaluated at runtime.

Notes



Example: ServerLib.dll can demand that the executing thread have an authenticated security principal within the BUILTIN\Administrators role for any methods that access the file system. This additional check can complement resource protection afforded by the Windows operating system's checks against the process identity.

The following is an example of a method that declaratively (using .NET attributes) demands the identity attached to the call context is within the BUILTIN\Administrators role:

```
<PrincipalPermission(SecurityAction.Demand,
 Role:="BUILTIN\Administrators")> _
Public Sub AdminOnlyMethod()
 'perform restricted actions
End Sub
```

For this demand to work the user must be authenticated, and its security principal attached to the executing thread. Later I'll review how this authentication step is handled by different application architectures. For now, just know that each thread has access to its security principal through the CurrentPrincipal property. WindowsPrincipal and GenericPrincipal types, and any custom principals also implementing the IPrincipal interface, can be attached to a thread by setting this property. A WindowsPrincipal is used for Windows authentication schemes, and a GenericPrincipal for custom authentication schemes that don't rely on the Windows domain.

Notes

The IPrincipal interface provides a consistent way for security principals to expose information about a user's identity and roles. This includes access to an identity object, represented at runtime by a valid IIdentity type for example a WindowsIdentity or GenericIdentity.

The point here is that the user is identified by an identity abstraction at runtime, and this identity must be somehow attached to each executing thread to be applicable during role-based security checks. Assuming the main thread has been assigned a valid security principal, any new thread will be initialized by the runtime with the same principal. That means you shouldn't need to manually propagate this information to new threads within the same application domain.

13.2 Authentication and Authorization

Authentication is the process that determines the identity of a user. After a user has been authenticated, a developer can determine if the identified user has authorization to proceed. It is impossible to give an entity authorization if no authentication process has been applied.

Authorization is the process of determining whether an authenticated user is permitted access to any part of an application, access to specific points of an application, or access only to specified datasets that the application provides. Authenticating and authorizing users and groups enable you to customize a site based on user types or preferences.

13.2.1 Applying Authentication Measures

ASP.NET provides many different types of authentication measures to use within your applications, including basic authentication, digest authentication, forms authentication, Passport, and Integrated Windows authentication. You also can develop your own authentication methods. You should never authorize access to resources you mean to be secure if you haven't applied an authentication process to the requests for the resources.

The different authentication modes are established through settings that can be applied to the application's web.config file or in conjunction with the application server's Internet Information Services (IIS) instance.

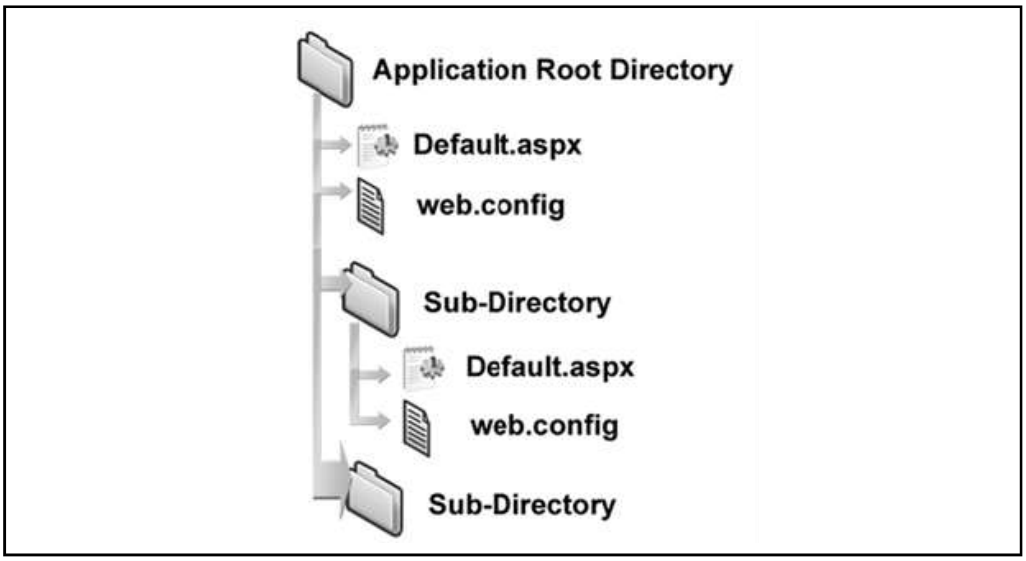
ASP.NET is configured through a series of .config files on the application server. These are XML-based files that enable you to easily change how ASP.NET behaves. This is an ideal way to work with the configuration settings you require. ASP.NET configuration files are applied in a hierarchal manner. The .NET Framework provides a server-level configuration file called the machine.config file, which can be found at

C:\Windows\Microsoft.NET\Framework\v2.0xxxxx\CONFIG. The folder contains machine.config and machine.config.comments files. These files provide ASP.NET application settings at a server-level, meaning that the settings are applied to each and every ASP.NET application that resides on the particular server.

Aweb.config file is another XML-based configuration file that resides in the root of the Web application. The settings applied in the web.config file override the same settings applied in the higher-level machine.config file.

You can even nest the web.config files so that the main application web.config file is located in the root directory of your application, but additional web.config files reside in some of the application's subdirectories. The web.config files contained in any of the subdirectories supersede the root directory's web.config file. Therefore, any settings applied through a subdirectory's web.config file change whatever was set in the application's main web.config file.

Notes



In a lot of the examples in this unit, you use the web.config file to apply the authentication and authorization mechanics you want in your applications. You also can work with IIS to apply settings directly to your applications.

13.2.2 The <authentication> Node

You use the <authentication> node in the application's web.config file to set the type of authentication your ASP.NET application requires:

```
<system.web>
<authentication mode="Windows|Forms|Passport|None">

</authentication>
</system.web>
```

The <authentication> node uses the mode attribute to set the form of authentication that is to be used. Options include Windows, Forms, Passport, and None. Each option is explained in the table.

Provider	Description
Windows	Windows authentication is used together with IIS authentication. Authentication is performed by IIS in the following ways: basic, digest, or Integrated Windows Authentication. When IIS authentication is complete, ASP.NET uses the authenticated identity to authorize access. This is the default setting.
Form	Requests that are not authenticated are redirected to an HTML form using HTTP client-side redirection. The user provides his login information and submits the form. If the application authenticates the request, the system issues a form that contains the credentials or a key for reacquiring the identity.
Passport	A centralized authentication service provided by Microsoft that offers single login and core profile services for member sites. This mode of authentication was de-emphasized by Microsoft at the end of 2004.
None	No authentication mode is in place with this setting.

As you can see, a couple of methods are at your disposal for building an authentication/ authorization model for your ASP.NET applications.

Notes

Task Programmer says ASP.NET programming is more secure than other type of programming? Specify.

13.3 Forms Authentication

Forms-based authentication is a popular mode of authenticating users to access an entire application or specific resources within an application. Using it enables you to put the login form directly in the application so that the end user simply enters his username and password into an HTML form contained within the browser itself. One negative aspect of forms-based authentication is that the usernames and passwords are sent as clear text unless you are using SSL.

It's easy and relatively straightforward to implement forms-based authentication in your Web application. To begin with, you make some modifications to your application's web.config file, as illustrated in Listing 13.1.

Listing 13.1: Modifying the web.config file for forms-based authentication

```
<system.web>
       <authentication mode="Forms">
              <forms name="Wrox" loginUrl="Login.aspx" path="/" />
       </authentication>

       <authorization>
              <deny users="?" />
       </authorization>
</system.web>
```

This is the structure you must apply to the web.config file. First, using the <authorization> element described earlier, you are denying access to the application to all anonymous users. Only authenticated users are allowed to access any page contained within the application.

If the requestor is not authenticated, what is defined in the <authentication> element is put into action. The value of the mode attribute is set to Forms to employ forms-based authentication for your Web application. The next attribute specified is loginUrl, which points to the page that contains the application's login form. In this example, Login.aspx is specified as a value. If the end user trying to access the application is not authenticated, his request is redirected to Login.aspx so that the user can be authenticated and authorized to proceed. After valid credentials have been provided, the user is returned to the location in the application where he originally made the request. The final attribute used here is path. It simply specifies the location in which to save the cookie used to persist the authorized user's access token. In most cases, you want to leave the value as /.

The table shows the describes each of the possible attributes for the <forms> element.

Notes

Attribute	Description
name	This is the name that is assigned to the cookie saved used to remember the user from request to request. The default value is .ASPXAUTH.
loginUrl	Specifies the URL to which the request is redirected for login if no valid authentication cookie is found. The default value is Login.aspx.
protection	Specifies the amount of protection you want to apply to the authentication cookie. The four available settings are: ● ***All***: The application uses both data validation and encryption to protect the cookie. This is the default setting. ● ***None***: Applies no encryption to the cookie. ● ***Encryption***: The cookie is encrypted but data validation isn't performed on it. Cookies used in this manner might be subject to plain text attacks. ● ***Validation***: The opposite of the Encryption setting. Data validation is performed, but the cookie is not encrypted.
path	Specifies the path for cookies issued by the application. In most cases you want to use /, which is the default setting.
timeout	Specifies the amount of time, in minutes, after which the cookie expires. The default value is 30.
cookieless	Specifies whether the forms-based authentication process should use cookies when working with the authentication/authorization process.
defaultUrl	Specifies the default URL.
domain	Specifies the domain name to be sent with forms authentication cookies.
slidingExpiration	Specifies whether to apply a sliding expiration to the cookie. If set to True, the expiration of the cookie is reset with each request made to the server. The default value is False.
enableCrossAppsRedirect	Specifies whether to allow for cross-application redirection.
requireSSL	Specifies whether a Secure Sockets Layer (SSL) connection is required when transmitting authentication information.

After the web.config file is in place, the next step is to create a typical page for your application that people can access. Listing 13.2 presents a simple page.

Listing 13.2: A simple page—Default.aspx

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

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
       <title>The Application</title>
</head>
<body>
       <form id="form1" runat="server">
       <div>
              Hello World
       </div>
       </form>
</body>
</html>
```

Notes

As you can see, this page simply writes Hello World to the browser. The real power of forms authentication is shown in the Login.aspx page presented in Listing 13.3.

Listing 13.3: The Login.aspx page

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

<script runat="server">
       Protected Sub Button1_Click(ByVal sender As Object, _
              ByVal e As System.EventArgs)
       If (TextBox1.Text = "BillEvjen" And TextBox2.Text = "Bubbles") Then
              FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, True)
       Else
              Response.Write("Invalid credentials")
       End If
End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
       <title>Login Page</title>
</head>
<body>
       <form id="form1" runat="server">
       <div>
       Username<br />
       <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
       <br/>
       Password<br />
       <asp:TextBox ID="TextBox2" runat="server"
       TextMode="Password"></asp:TextBox><br />
       <br/>
       <asp:Button ID="Button1" OnClick="Button1_Click" runat="server"
       Text="Submit" />
       </div>
</form>
</body>
</html>
```

Login.aspx has two simple TextBox controls and a Button control that ask the user to submit his username and password. The Button1_Click event uses the RedirectFromLoginPage method of the FormsAuthentication class. This method does exactly what its name implies it redirects the request from Login.aspx to the original requested resource.

RedirectFromLoginPage takes two arguments. The first is the name of the user, used for cookie authentication purposes. This argument doesn't actually map to an account name and is used by ASP.NET's URL authorization capabilities. The second argument specifies whether a durable cookie should be issued. If set to True, the end user does not need to log in again to the application from one browser session to the next.

Notes

Using the three pages you've constructed, each request for the Default.aspx page from Listing 13.2 causes ASP.NET to check that the proper authentication token is in place. If the proper token is not found, the request is directed to the specified login page (in this example, Login.aspx). Looking at the URL in the browser, you can see that ASP.NET is using a querystring value to remember where to return the user after he has been authorized to proceed:

```
http://localhost:35089/Security/
Login.aspx?ReturnUrl=%2fSecurity%2fDefault.aspx
```

Here the querystring ReturnUrl is used with a value of the folder and page that was the initial request.

Look more closely at the Login.aspx page from Listing 13.3, and note that the values placed in the two text boxes are checked to make sure they abide by a specific username and password. If they do, the RedirectFromLoginPage method is invoked; otherwise, the Response.Write statement is used. In most cases, you don't want to hardcode a username and password in your code. Many other options exist for checking whether usernames and passwords come from authorized users. Some of the other options follow.

13.3.1 Authenticating against Values contained in the web.config File

The previous example is not the best approach for dealing with usernames and passwords offered for authentication. It is never a good idea to hardcode these things directly into your applications. Take a quick look at storing these values in the web.config file itself.

The <forms> element in web.config that you worked with in Listing 13.1 can also take a sub-element. The sub-element, <credentials>, allows you to specify username and password combinations directly in the web.config file. You can choose from a couple of ways to add these values. The simplest method is shown in Listing 13.4.

Listing 13.4: Modifying the web.config file to add username/password values

```
<system.web>
	<authentication mode="Forms">
		<forms name="Wrox" loginUrl="Login.aspx" path="/">
			<credentials passwordFormat="Clear">
				<user name="BillEvjen" password="Bubbles" />
			</credentials>
		</forms>
	</authentication>

	<authorization>
		<deny users="?" />
	</authorization>
</system.web>
```

The <credentials> element has been included to add users and their passwords to the configuration file. <credentials> takes a single attribute passwordFormat. The possible values of passwordFormat are Clear, MD5, and SHA1. The following list describes each of these options:

1. ***Clear:*** Passwords are stored in clear text. The user password is compared directly to this value without further transformation.

Notes

2. ***MD5:*** Passwords are stored using a Message Digest 5 (MD5) hash digest. When credentials are validated, the user password is hashed using the MD5 algorithm and compared for equality with this value. The clear-text password is never stored or compared. This algorithm produces better performance than SHA1.
3. ***SHA1:*** Passwords are stored using the SHA1 hash digest. When credentials are validated, the user password is hashed using the SHA1 algorithm and compared for equality with this value. The clear-text password is never stored or compared. Use this algorithm for best security.

In the example from Listing 13.4, you use a setting of Clear. This isn't the most secure method, but it is used for demonstration purposes. A sub-element of <credentials> is <user>; that's where you define the username and password for the authorized user with the attributes name and password.

The next step is to change the Button1_Click event on the Login.aspx page shown earlier. This is illustrated in Listing 13.5.

Listing 13.5: Changing the Login.aspx page to work with the web.config file

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


<script runat="server">
       Protected Sub Button1_Click(ByVal sender As Object, _
              ByVal e As System.EventArgs)
       If FormsAuthentication.Authenticate(TextBox1.Text, TextBox2.Text) Then
       FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, True)

              Else
                     Response.Write("Invalid credentials")
              End If
       End Sub
</script>
```

In this example, you simply use the Authenticate method to get your ASP.NET page to look at the credentials stored in the web.config file for verification. The Authenticate method takes two parameters the username and the password that you are passing in to be checked. If the credential lookup is successful, the RedirectFromLoginPage method is invoked.

It is best not to store your users' passwords in the web.config file as clear text as the preceding example did. Instead, use one of the available hashing capabilities so you can keep the end user's password out of sight of prying eyes. To do this, simply store the hashed password in the configuration file as shown in Listing 13.6.

Listing 13.6: Using encrypted passwords

```
<forms name="Wrox" loginUrl="Login.aspx" path="/">
       <credentials passwordFormat="SHA1">
              <user name="BillEvjen"
              password="58356FB4CAC0B801F011B397F9DFF45ADB863892" />
       </credentials>
</forms>
```

Notes

Using this kind of construct makes it impossible for even the developer to discover a password because the clear text password is never used. The Authenticate method in the Login.aspx page hashes the password using SHA1 (because it is the method specified in the web.config's <credentials> node) and compares the two hashes for a match. If a match is found, the user is authorized to proceed.

When using SHA1 or MD5, the only changes you make are in the web.config file and nowhere else. You don't have to make any changes to the login page or to any other page in the application. To store hashed passwords, however, you use the Forms Authentication. Hash Password For Storing In ConfigFile method (probably the longest method name in the .NET Framework). You accomplish this in the following manner:

FormsAuthentication.HashPasswordForStoringInConfigFile(TextBox2.Text, "SHA1")

13.3.2 Authenticating against Values in a Database

Another common way to retrieve username/password combinations is by getting them directly from a datastore of some kind. This enables you, for example, to check the credentials input by a user against values stored in Microsoft's SQL Server. The code for this is presented in Listing 13.7.

Listing 13.7: Checking credentials in SQL Server (Login.aspx)

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<script runat="server">
       Protected Sub Button1_Click(ByVal sender As Object, _
              ByVal e As System.EventArgs)

              Dim conn As SqlConnection
              Dim cmd As SqlCommand
              Dim cmdString As String = "SELECT [Password] FROM [AccessTable]
              WHERE" & _ " (([Username] = @Username) AND ([Password] =
              @Password))"

              conn = New SqlConnection("Data Source=localhost;Initial " & _
              "Catalog=Northwind;Persist Security Info=True;User ID=sa")
              cmd = New SqlCommand(cmdString, conn)

              cmd.Parameters.Add("@Username", SqlDbType.VarChar, 50)
              cmd.Parameters("@Username").Value = TextBox1.Text
              cmd.Parameters.Add("@Password", SqlDbType.VarChar, 50)
              cmd.Parameters("@Password").Value = TextBox2.Text

              conn.Open()

              Dim myReader As SqlDataReader
```

Notes

```
        myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection)
        If myReader.Read() Then
        FormsAuthentication.RedirectFromLoginPage(TextBox1.Text,
        False)
        Else
             Response.Write("Invalid credentials")
        End If
        myReader.Close()
    End Sub
</script>
```

Leave everything else from the previous examples the same, except for the Login.aspx page. You can now authenticate usernames and passwords against data stored in SQL Server. In the Button1_Click event, a connection is made to SQL Server. (For security reasons, you should store your connection string in the web.config file.) Two parameters are passed in—the inputs from TextBox1 and TextBox2. If a result is returned, the RedirectFromLoginPage method is invoked.

13.3.3 Using the Login Control with Forms Authentication

You have seen how to use ASP.NET forms authentication with standard ASP.NET server controls, such as simple TextBox and Button controls. You can also use the latest ASP.NET 2.0 server controls such as the new Login server control with your custom-developed forms-authentication framework instead of using other controls. This really shows the power of ASP.NET you can combine so many pieces to construct the solution you want.

Listing 13.8 shows a modified Login.aspx page using the new Login server control.

Listing 13.8: Using the Login server control on the Login.aspx page

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

<script runat="server">
    Protected Sub Login1_Authenticate(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.AuthenticateEventArgs)

    If (Login1.UserName = "BillEvjen" And Login1.Password = "Bubbles") Then
    FormsAuthentication.RedirectFromLoginPage(Login1.UserName, _
    Login1.RememberMeSet)
Else
    Response.Write("Invalid credentials")
End If
End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Login Page</title>
</head>
<body>
```

Notes

```
        <form id="form1" runat="server">
        <div>
        <asp:Login ID="Login1" runat="server"
        OnAuthenticate="Login1_Authenticate">
        </asp:Login>
        </div>
        </form>
</body>
</html>
```

Because no Button server control is on the page, you use the Login control's OnAuthenticate attribute to point to the authentication server-side event Login1_Authenticate. The event takes care of the authorization lookup (although the values are hardcoded in this example). The username text box of the Login control can be accessed via the Login1.UserName declaration, and the password can be accessed using Login1.Password. The Login1.RememberMeSet property is used to specify whether to persist the authentication cookie for the user so that he is remembered on his next visit.

This example is a bit simpler than creating your own login form using TextBox and Button controls. You can give the Login control a predefined look-and-feel that is provided for you. You can also get at the subcontrol properties of the Login control a bit more easily. In the end, it really is up to you as to what methods you employ in your ASP.NET applications.

13.3.4 Forms Authentication Class

As you can tell from the various examples in the forms authentication part of this unit, a lot of what goes on depends on the FormsAuthentication class itself. For this reason, you should learn what that class is all about.

FormsAuthentication provides a number of methods and properties that enable you to read and control the authentication cookie as well as other information (such as the return URL of the request). The table details some of the methods and properties available in the FormsAuthentictation class.

Method/Property	Description
Authenticate	This method is used to authenticate credentials that are stored in a configuration file (such as the web.config file)
Decrypt	Returns an instance of a valid, encrypted authentication ticket retrieved from an HTTP cookie as an instance of a FormsAuthenticationTicket class.
Encrypt	Creates a string which contains a valid encrypted authentication ticket that can be used in an HTTP cookie.
FormsCookieName	Returns the name of the cookie for the current application.
FormsCookiePath	Returns the cookie path (the location of the cookie) for the current application.
GetAuthCookie	Provides an authentication cookie for a specified user.
GetRedirectUrl	Returns the URL to which the user is redirected after being authorized by the login page.
HashPasswordFor Scoring InConfigFile	Creates a hash of a provided string password. This method takes two parameters—one is the password and the other is the type of hash to perform on the string. Possible hash values include SHA1 and MD5.

Contd...

Notes

Initialize	Performs an initialization of the FormsAuthentication class by reading the configuration settings in the web. config file, as well as getting the cookies and encryption keys used in the given instance of the application.
RedirectFromLoginPage	Performs a redirection of the HTTP request back to the original requested page. This should be performed only after the user has been authorized to proceed.
RenewTicketIfOld	Conditionally updates the sliding expiration on a FormsAuthenticationTicket instance.
RequireSSL	Specifies whether the cookie should be transported via SSL only (HTTPS).
SetAuthCookie	Creates an authentication ticket and attaches it to a cookie that is contained in the outgoing response.
SignOut	Removes the authentication ticket.
SlidingExpiration	Provides a Boolean value indicating whether sliding expiration is enabled.

13.4 Windows Authentication

Windows-based authentication is handled between the Windows server where the ASP.NET application resides and the client machine. In a Windows-based authentication model, the requests go directly to IIS to provide the authentication process. This type of authentication is quite useful in an intranet environment where you can let the server deal completely with the authentication process especially in environments where users are already logged onto a network. In this scenario, you simply grab and utilize the credentials that are already in place for the authorization process.

IIS first takes the user's credentials from the domain login. If this process fails, IIS displays a pop-up a dialog box so the user can enter or re-enter his login information. To set up your ASP.NET application to work with Windows-based authentication, begin by creating some users and groups.

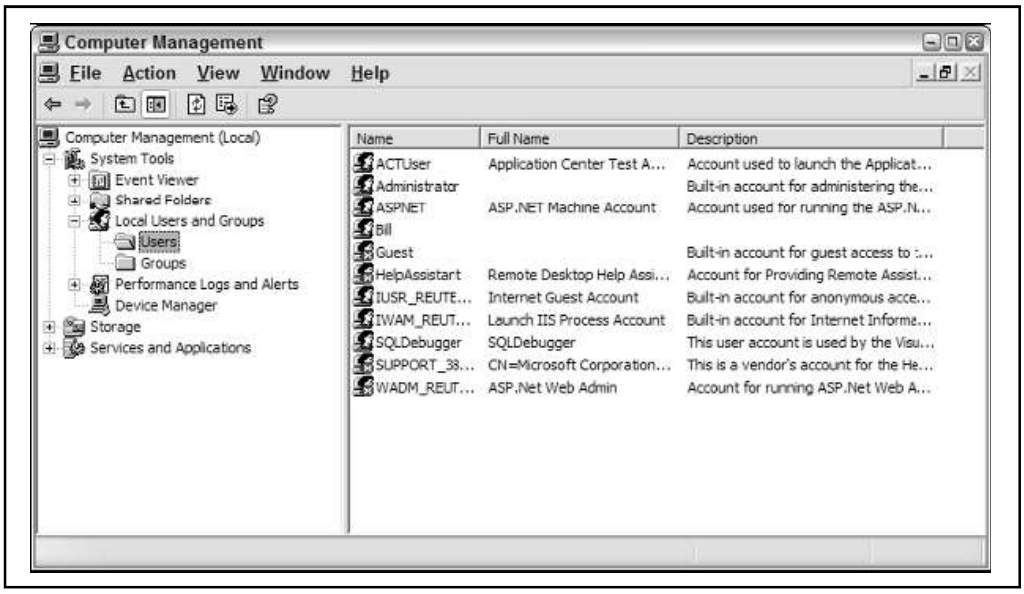
13.4.1 Creating Users

You use aspects of Windows-based authentication to allow specific users who have provided a domain login to access your application or parts of your application. Because it can use this type of authentication, ASP.NET makes it quite easy to work with applications that are deployed in an intranet environment. If a user has logged onto a local computer as a domain user, he won't need to be authenticated again when accessing a network computer in that domain.

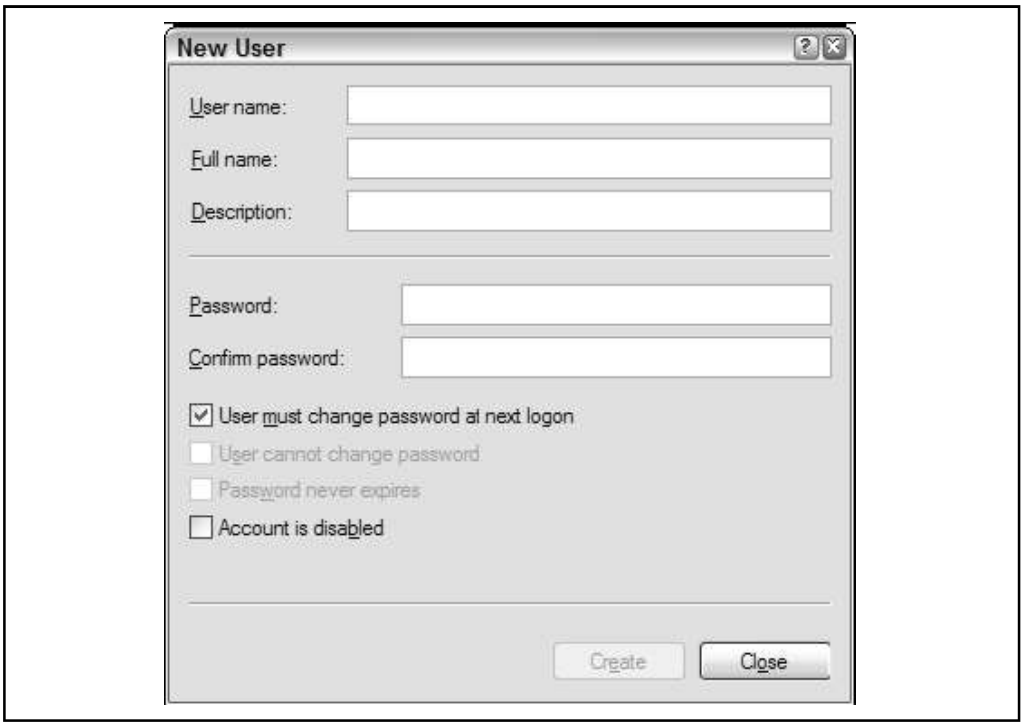
The following steps show you how to create a user. It is important to note that you must have sufficient rights to be authorized to create users on a server. If you are authorized, the steps to create users are as follows:

1. Within your Windows XP or Windows Server 2003 server, choose Start → Control Panel → Administrative Tools → Computer Management. The Computer Management utility opens. It manages and controls resources on the local Web server. You can accomplish many things using this utility, but the focus here is on the creation of users.
2. Expand the System Tools node.
3. Expand the Local Users and Groups node.
4. Select the Users folder. You see something similar to the results shown in figure.

Notes

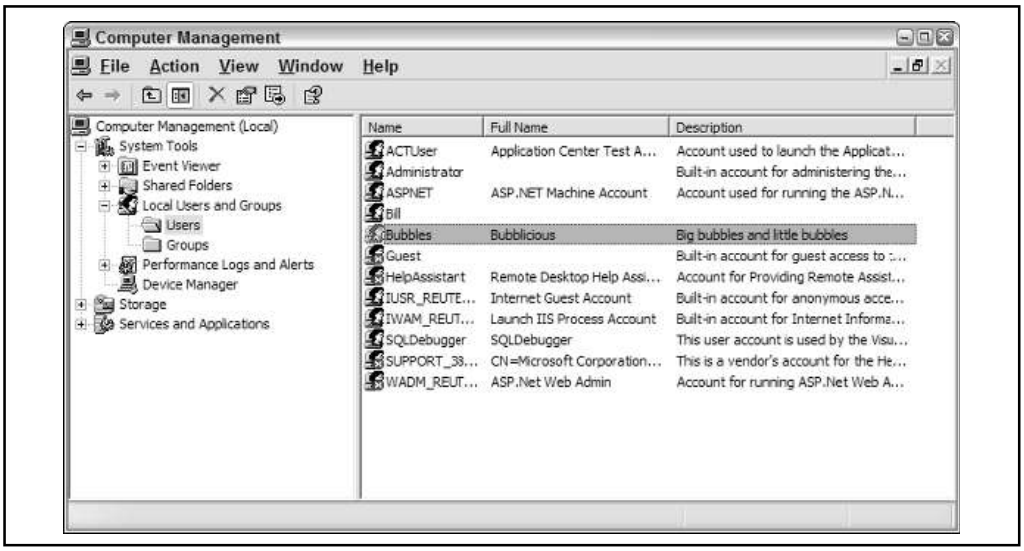


5. Right-click the Users folder and select New User. The New User dialog appears, as shown in figure.



6. Give the user a name, password, and description stating that this is a test user. This example calls the user Bubbles.
7. Uncheck the check box that requires the user to change his password at the next login.
8. Click the Create button. Your test user is created and presented in the Users folder of the Computer Management utility, as shown in figure.

Notes



Now create a page to work with this user.

13.4.2 Authenticating and Authorizing a User

Now create an application that enables the user to enter it. You work with the application's web.config file to control which users are allowed to access the site and which users are not allowed.

Add the section presented in Listing 13.9 to your web.config file.

Listing 13.9: Denying all users through the web.config file

```
<system.web>
<authentication mode="Windows" />
<authorization>
<deny users="*" />
</authorization>
</system.web>
```

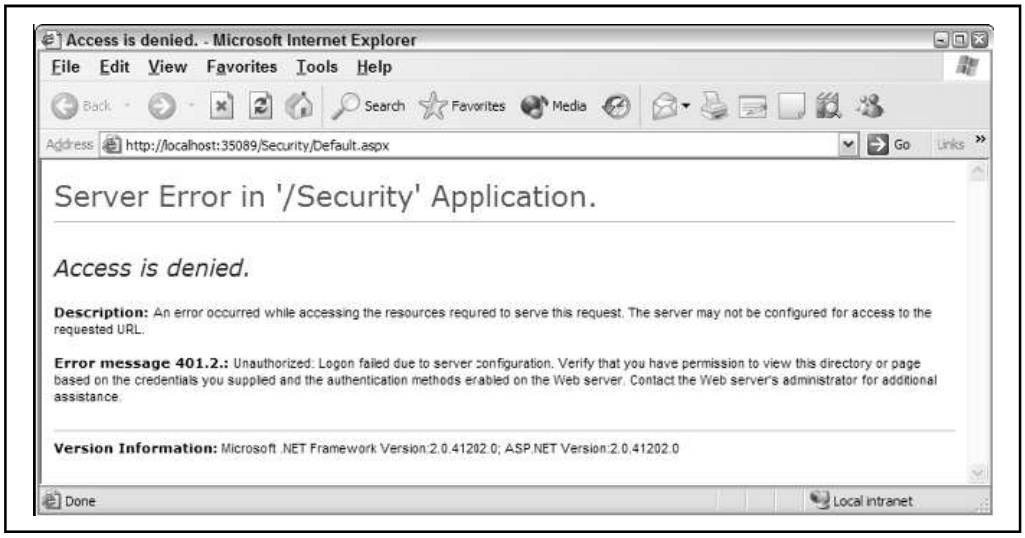
In this example, the web.config file is configuring the application to employ Windows-based authentication through the use of the <authentication> element's mode attribute. In addition, the <authorization> element is used to define specifics about the users or groups who are permitted access to the application. In this case, the <deny> element specifies that all users (even if they are authenticated) are denied access to the application. Not permitting specific users with the <allow> element doesn't make much sense, but for this example, leave it like it is. The results are illustrated in figure.

Any end user authenticated or not who tries to access the site sees a large "Access is denied" statement in his browser window, which is just what you want for those not allowed to access your application!

In most instances, however, you want to allow at least some users to access your application. Use the <allow> element in the web.config file to allow a specific user. Here's the syntax:

```
<allow users="Domain\Username"/>
```

Notes



Listing 13.10 shows how the user is permitted access.

Listing 13.10: Allowing a single user through the web.config file

```
<system.web>
<authentication mode="Windows" />
<authorization>
<allow users="REUTERS-EVJEN\Bubbles" />
<deny users="*"/>
</authorization>
</system.web>
```

Even though all users (even authenticated ones) are denied access through the use of the <deny> element, the definitions defined in the <allow> element take precedence. In this example, a single user Bubbles is allowed.

Now, if you are logged on to the client machine as the user Bubbles and run the page in the browser, you get access to the application.

`Looking Closely at the <allow> and <deny> Nodes:`

The <allow> and <deny> nodes enable you to work not only with specific users, but also with groups.

The elements support the attributes defined in the table.

Attribute	Description
users	Enables you to specify users by their domain and/or name.
roles	Enables you to specify access groups that are allowed or denied access.
verbs	Enables you to specify the HTTP transmission method that is allowed or denied access.

Notes

When using any of these attributes, you can specify all users with the use of the asterisk (*):

```
<allow roles="*" />
```

In this example, all roles are allowed access to the application. Another symbol you can use with these attributes is the question mark (?), which represents all anonymous users. For example, if you want to block all anonymous users from your application, use the following construction:

```
<deny users="?" />
```

When using users, roles, or verbs attributes with the <allow> or <deny> elements, you can specify multiple entries by separating the values with a comma. If you are going to allow more than one user, you can either separate these users into different elements as shown here

```
<allow users="MyDomain\User1" />
<allow users="MyDomain\User2" />
```

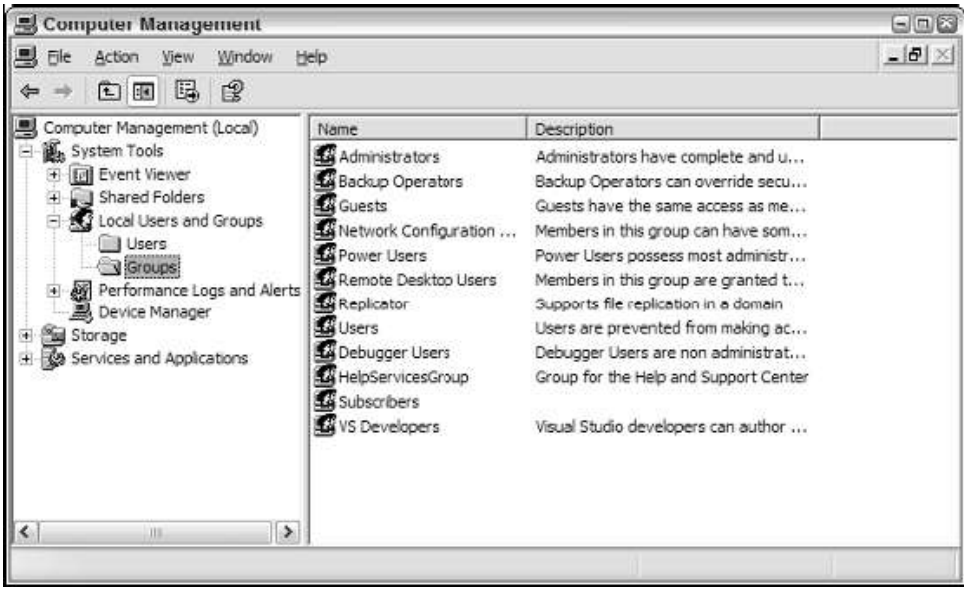
or you can use the following:

```
<allow users="MyDomain\User1, MyDomain\User2" />
```

Use the same construction when defining multiple roles and verbs.

Task Explain step-by-step how will you create new user in windows authentication.

13.4.3 Authenticating and Authorizing a Group



You can define groups of individuals allowed or denied access to your application or the application's resources. Your server can contain a number of different groups, each of which can have any number of users belonging to it. It's also possible for a single user to belong to

Notes

multiple groups. Pull up the Computer Management utility (Start → Control Panel → Administrative Tools → Computer Management) to access the list of the groups defined on the server you are working with. Simply click the Groups folder in the Computer Management utility, and the list of groups is displayed, as illustrated in figure.

Right-click in the Groups folder to select New Group. The New Group dialog displays.

To create a group, give it a name and description; then click the Add button and select the users whom you want to be a part of the group. After a group is created, you can allow it access to your application like this:

```
<allow roles="MyGroup" />
```

You can use the roles attribute in either the <allow> or <deny> element to work with a group that you have created or with a specific group that already exists.

13.4.4 Authenticating and Authorizing an HTTP Transmission Method

In addition to authenticating and authorizing specific users or groups of users, you can also authorize or deny requests that come via a specific HTTP transmission protocol. This is done using the verb attribute in the <allow> and <deny> elements.

```
<deny verbs="GET, DEBUG"/>
```

In this example, requests that come in using the HTTP GET or HTTP DEBUG protocols are denied access to the site. Possible values for the verbs attribute include POST, GET, HEAD, and DEBUG.

13.4.5 Integrated Windows Authentication

So far, you've been using the default Integrated Windows authentication mode for the authentication/authorization process. This is fine if you are working with an intranet application and each of the clients is using Windows, the only system that the authentication method

Notes

supports. This system of authentication also requires the client to be using Microsoft's Internet Explorer, which might not always be possible.

Integrated Windows authentication was previously known as NTLM or Windows NT Challenge/ Response authentication. This authentication model has the client prove its identity by sending a hash of its credentials to the server that is hosting the ASP.NET application. Along with Microsoft's Active Directory, a client can also use Kerberos if it's using Microsoft's Internet Explorer 5 or higher.

13.4.6 Basic Authentication

Another option is to use Basic authentication, which also requires a username and password from the client for authentication. The big plus about Basic authentication is that it is part of the HTTP specification and therefore is supported by most browsers. The negative aspect of Basic authentication is that it passes the username and password to the server as clear text, meaning that the username and password are quite visible to prying eyes. For this reason, it is important to use Basic authentication along with SSL (Secure Sockets Layer).

To implement Basic authentication for your application, you must pull up IIS and open the Properties dialog for the Web site you are working with. Select the Directory Security tab and click the Edit button in the Anonymous Access and Authentication Control box. The Authentication Methods dialog box opens.

Uncheck the Integrated Windows Authentication check box at the bottom and check the Basic Authentication check box above it. When you do, you are warned that this method transmits usernames and passwords as clear text.

Notes

End by clicking OK in the dialog. Now your application uses Basic authentication instead of Integrated Windows authentication.

13.4.7 Digest Authentication

Digest authentication is the final mode you explore in this unit. The model alleviates the Basic authentication problem of passing the client's credentials as clear text. Instead, Digest authentication uses an algorithm to encrypt the client's credentials before they are sent to the application server.

To use Digest authentication, you are required to have a Windows domain controller. One of the main issues that arises with Digest authentication is that it is not supported on all platforms and requires browsers that conform to the HTTP 1.1 specification. Digest authentication, however, not only works well with firewalls, but it is also compatible with proxy servers.

You can select Digest authentication as the choice for your application in the same Authentication Methods dialog simply select the Digest Authentication check box.

Case Study

Security Loopholes

Utpal had just joined SystemX as Systems Manager. But he was a worried man looking at the current state of affairs at SystemX. As a part of assessing hardware and software requirements, it was found that out of the 364 desktops at the corporate office; more than half did not have their antivirus software updated with recent virus signature files. Three - fourths had not changed the default e-mail password (it was the user name) and no one had installed OS patches. And one of its local mail servers seemed to be an open relay! For a fleeting moment, he wondered about the situation at the seven branch offices across the country.

SystemX used the Net extensively in dealing with its branches, customers and suppliers. Information like contract documents, marketing plans, Cheque and Draft numbers, bank account details and collection details were regularly transmitted by e-mail. Utpal's first thought was that he would recommend that SystemX bring in a security consultant. But the budget constraints meant that his recommendation was unlikely to find favour. He was beginning to feel a bit out of depth and was wondering what he should do to ensure that SystemX's data remained safe and secure.

Questions

1. What security loopholes come to the fore in the situation described? How can these be plugged?
2. What is the importance of a "security budget" in the context of the given situation?

13.5 Summary

- In this unit, you learned about the multilayered security architecture in ASP.NET and IIS and how you can safeguard your web pages and web services by using a custom login page or Windows authentication. You also learned the basics about certificates and SSL.
- The first goal of this unit was to introduce you to the various security components supplied by the .NET 2.0 base class libraries.

Notes

- As you have seen, there are numerous security-centric namespaces, most of which have a direct impact on ASP.NET web applications.
- Recall that the framework provides numerous types to work with standard encryption atoms (hash codes, asymmetric/symmetric encryption) and traditional role-based security.
- The remainder of this unit focused exclusively upon the ASP.NET security framework.
- We began by reviewing the core Forms authentication model, which has been present since the inception of the .NET platform. Once we established the basics, we examined how the Membership type and various server controls can be used to simplify the authentication process. Next, we revisited the notion of role-based security within the context of the Roles API.

13.6 Keywords

Application Domain (AppDomain): A boundary that the common language runtime establishes around objects created within the same application scope (that is, anywhere along the sequence of object activations beginning with the application entry point). Application domains help isolate objects created in one application from those created in other applications so that run-time behavior is predictable.

Authentication: In .NET Framework security, the process of discovering and verifying the identity of a principal by examining the user's credentials against some authority.

Authorization: In .NET Framework security, the process of limiting access rights by granting or denying specific permissions to an authenticated identity or principal.

Site Navigation: In ASP.NET Web sites, the process of displaying controls such as menus, a tree view, or SiteMapPath (breadcrumb) controls that assist users in finding pages of interest. Site navigation is typically driven from a sitemap.

13.7 Self Assessment

Fill in the blanks:

1. ..... is the process that determines the identity of a user.
2. ..... is the process of determining whether an authenticated user is permitted access to any part of an application.
3. IIS stands for .....
4. ..... authentication is a popular mode of authenticating users to access an entire application or specific resources within an application.
5. Integrated Windows authentication was previously known as .....

State whether the following statements are True or False:

6. ASP.NET is configured through a series of .config files on the application server.
7. Aweb.config file is not XML-based configuration file that resides in the root of the Web application.
8. Possible values of passwordFormat are Clear, MD5, and SHA1.
9. For use Digest authentication, you are required to have a Windows domain controller.
10. Authenticating and authorizing users and groups enable you to customize a site based on user types or preferences.

Notes

13.8 Review Questions

1. What do you need to do to enable user profiles in your site?
2. What modifications do you need to make to web.config to specify what profile information you want to retain for your users?
3. What property of the Profile object do you use to determine whether a user is logged in?
4. How do you retain profile information that's not saved as a string?
5. How do you allow for profile information for a user without logging in?
6. How do you indicate that a specific piece of profile information should be retained for an anonymous user?
7. When a user logs in, how do you transfer any personalization data that the user might have entered as an anonymous user?
8. What's the difference between style sheet themes and customization themes?
9. Where do you specify settings for a skin?
10. How do you specify what theme to use on a page?

Answers: Self Assessment

1.	Authentication	2.	Authorization
3.	Internet Information Services	4.	Forms-based
5.	NTLM	6.	True
7.	False	8.	True
9.	True	10.	True

13.9 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

Notes

Unit 14: Deploying Website

CONTENTS

Objectives
Introduction
14.1 Visual Studio Projects
14.1.1 HTTP Project
14.1.2 FTP Project
14.1.3 File System Project
14.2 How to Deploy the Website?
14.2.1 Pre-compiling for Performance
14.2.2 Pre-compiling for Deployment
14.2.3 Creating a Website Installer
14.3 Deployment of a Website on IIS
14.3.1 IIS
14.3.2 Managing a Virtual Directory
14.3.3 Managing a Application Pool (Worker Processes)
14.3.4 Deploying Application
14.4 Summary
14.5 Keywords
14.6 Self Assessment
14.7 Review Questions
14.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe visual studio projects
- Explain how to deploy the website
- Know deploy website on IIS

Introduction

Several times we encounter a great and sophisticated web or desktop application that does not poses it's appropriate market share just for the reason of poorly written SETUP package and poorly designed deployment strategy. The fact is simple: If your users are not able to easily deploy your application then whatever sophistication or features you provide, they will be unable to even experience your application from the very first place.

Notes

After building a feature-rich application that streamlines your company operations or drives customers to your business, you need to be able to manage it effectively, and deploy it. That's the topic of this unit how the various Visual Studio models affect your deployment strategy.

14.1 Visual Studio Projects

Visual Studio 2005 gives you several options when building a Web site project (as opposed to earlier versions that depended upon IIS). These project models include the HTTP project, the FTP project, and the file project. Here's a summary of how each model works.

14.1.1 HTTP Project

The HTTP project is most like the model built into earlier versions of Visual Studio. Under the HTTP project model, Visual Studio creates a virtual directory under IIS and uses IIS to intercept requests during development time. Under this model, the solution file (the .sln file) resides in a directory specified under Visual Studio's project settings directory. The source code for the project is stored in the IIS virtual directory (that is, \Inetpub\wwwroot).

You may either have Visual Studio create a virtual directory for you, or you may create a virtual directory ahead of time. You may store the code for your Web site in any folder. The virtual directory just needs to point to that location.

Use this option if you want to work as closely as possible in the same mode as earlier versions of Visual Studio. In addition, using an IIS Web site during development lets you test the entire request path (not just the path through ASP.NET). This is important if you want to test an application that leverages IIS security, requires ISAPI filters, application pooling, or some other specific IIS features to run effectively. One reason to create a local Web site is to test your application against a local version of IIS. Using IIS as part of the development environment makes it easier to test these things. Of course, the downside to this approach is that you require IIS to be installed on your machine (it's not installed automatically you have to take a deliberate step to install it). Having IIS on your machine may also compromise security.

14.1.2 FTP Project

The FTP project is meant for those projects you want to manage remotely through an FTP server. For example, this is a good option if you use a remote hosting service to host your Web site. The FTP site option represents a reasonable means of getting files from your development environment to the hosting site.

When creating this type of site, Visual Studio will connect to any FTP server for which you have reading and writing privileges. You then use Visual Studio to manage the content on the remote FTP server.

You might use this option to test the Web site on the live-deployed server where it will actually be deployed.

14.1.3 File System Project

The file project is probably the most developer-oriented project. File System projects rely upon the Web server inside Visual Studio instead of IIS. When you specify a file system Web site, you may tell Visual Studio to put it anywhere on your file system or in a shared folder on another computer.

Notes

If you don't have access to IIS, or you don't have administration rights to the system on which you're developing, then you'll want to create a file system-based Web site project. The site runs locally, but independently of IIS. The most common scenario in this case is to develop and test a Web site on the file system. Then when it comes time to expose your site, simply create an IIS virtual directory and point it to the pages in the file system Web site.

By default, Visual Studio does not pre-compile your Web application. Once you've developed a site using Visual Studio, you may decide to pre-compile it.

14.2 How to Deploy the Website?

Earlier versions of Visual Studio (for example, Visual Studio 2003) automatically built ASP.NET applications when you hit the Build | Build Solution menu item. All the source code (the VB and the CS files) was compiled into a resulting assembly named the same as the project. This precompiled assembly went into the project's \bin directory and became part of the files used for deployment. ASP.NET will still pre-compile an application for you. However, now you have two choices as far as recompiling goes using a virtual path (for applications already defined in IIS) and a physical path (for sites that live on the file system). In addition, you must be deliberate about pre-compiling. The two pre-compiling options include (1) pre-compiling for performance and (2) pre-compiling for deployment. Pre-compiling a Web site involves using command line tools.

14.2.1 Pre-compiling for Performance

The first option is also known as "pre-compiling in place." This is useful for existing sites for which you want to enhance performance. When you pre-compile the source code behind your site, the primary benefit is that ASP.NET doesn't have to run that first compilation when the site is hit for the first time. If your site requires frequent updates to the code base, you may see a small amount of performance improvement.

To pre-compile an IIS-based site in place, open a Visual Studio command window. Navigate to the .NET directory on your machine (probably Windows\ Microsoft.Net\ Framework\ <versionnumber>). In that directory is a program named aspnet_compiler. Execute the aspnet_compiler program, with the name of the Web site as known by IIS following the –v switch. For example, if IIS has a virtual directory named MySite, the following command line will build it. The precompiled application ends up in the Temporary ASP.NET Files directory under your current .NET directory.

```
aspnet_compiler -v MySite
```

If the Web site is a file system Web site without an IIS virtual directory, use the –p command line parameter to specify the physical path.

This compilation option pre-compiles the code and places it in the bin directory for the application.

14.2.2 Pre-compiling for Deployment

Compiling for deployment involves compiling the code for a site and directing the output to a special directory from which it may be copied to the deployment machine or used in an install project (as we'll see momentarily). In this case, the compiler produces assemblies from all ASP.NET source files that are normally compiled at run time. That includes the code for the pages, source code within the App_Code directory, and resource files.

Notes

To pre-compile a site for deployment, open a Visual Studio command window. Navigate to the .NET directory. Run the aspnet_compiler command line program, specifying the source as either a virtual path or physical path. Provide the target folder following the input directory. For example, the following command builds the code in the MySite virtual directory and puts in c:\MySiteTarget.

```
aspnet_compiler -v MySite c:\MySiteTarget
```

If you add a –u command line parameter at the end of the command line, the compiler will compile some of the code and leave the page code files to be compiled just in time.

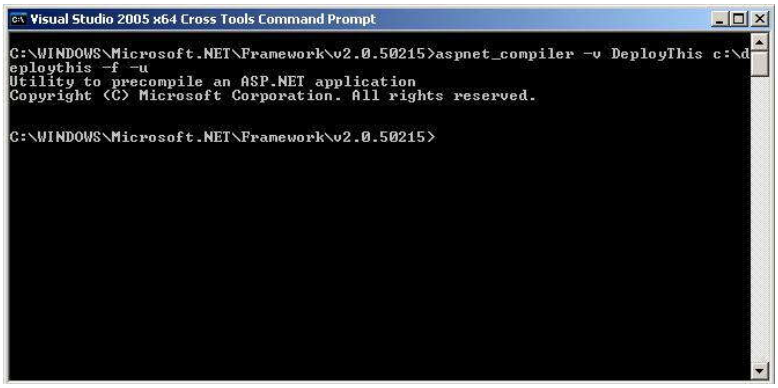
Once the code is compiled, one of the options you have is to build a Web setup program. The following example illustrates creating a Web setup program.

14.2.3 Creating a Website Installer

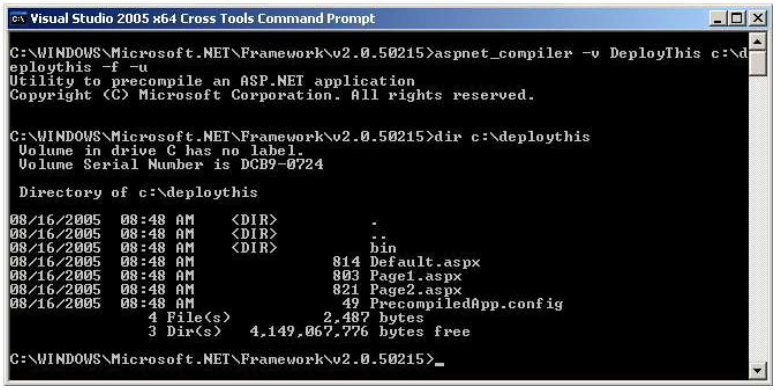
1. Start by creating a new site. Make it an HTTP site. Name the site DeployThis.
2. Create some content for the site.

Example: Add a few pages to the site, or borrow content from an earlier example.

3. Pre-compile the site for deployment. Tell the aspnet_compiler to use the Deploy this virtual directory as the source and to direct it to a target holding directory. The following graphic illustrates the command line. Use the –u option at the end of the command line to instruct the compiler to make an updateable Web site.

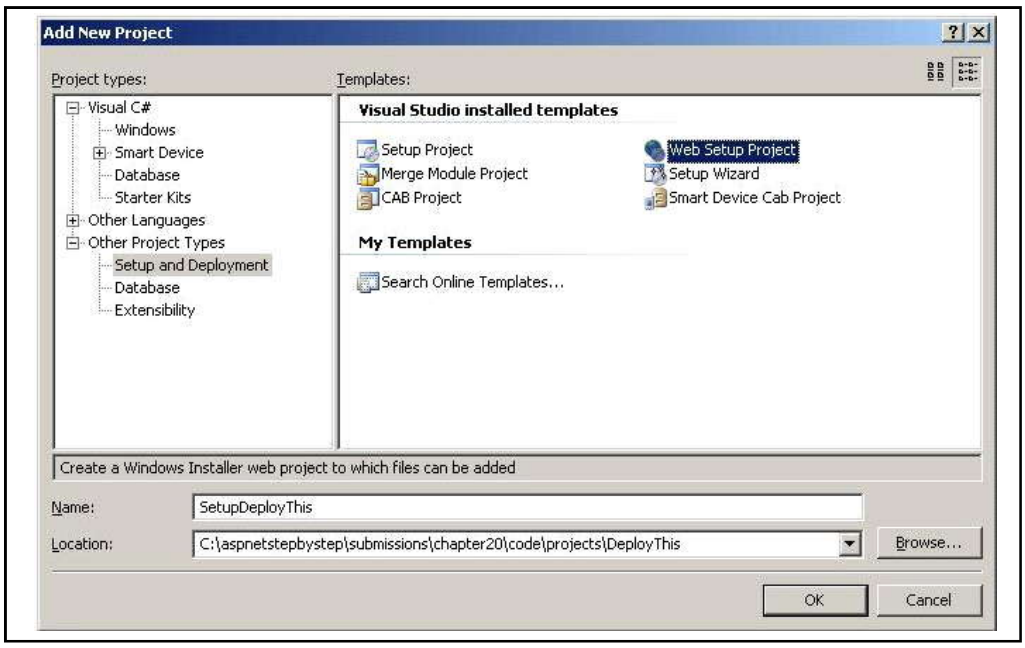


4. After the compiler runs, you'll have a target directory full of compiled code. The following graphic illustrates the results of the compilation.

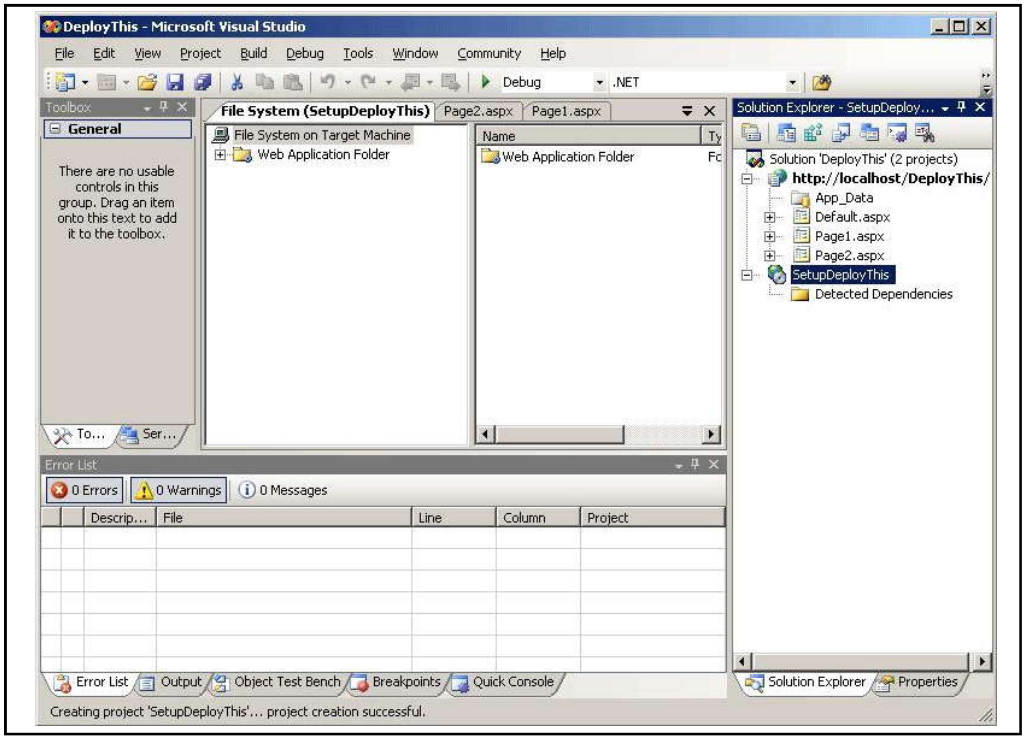


Notes

5. Add a second project to the solution. Make it a Web Setup Project, as shown in the following graphic. Name the project SetupDeployThis.

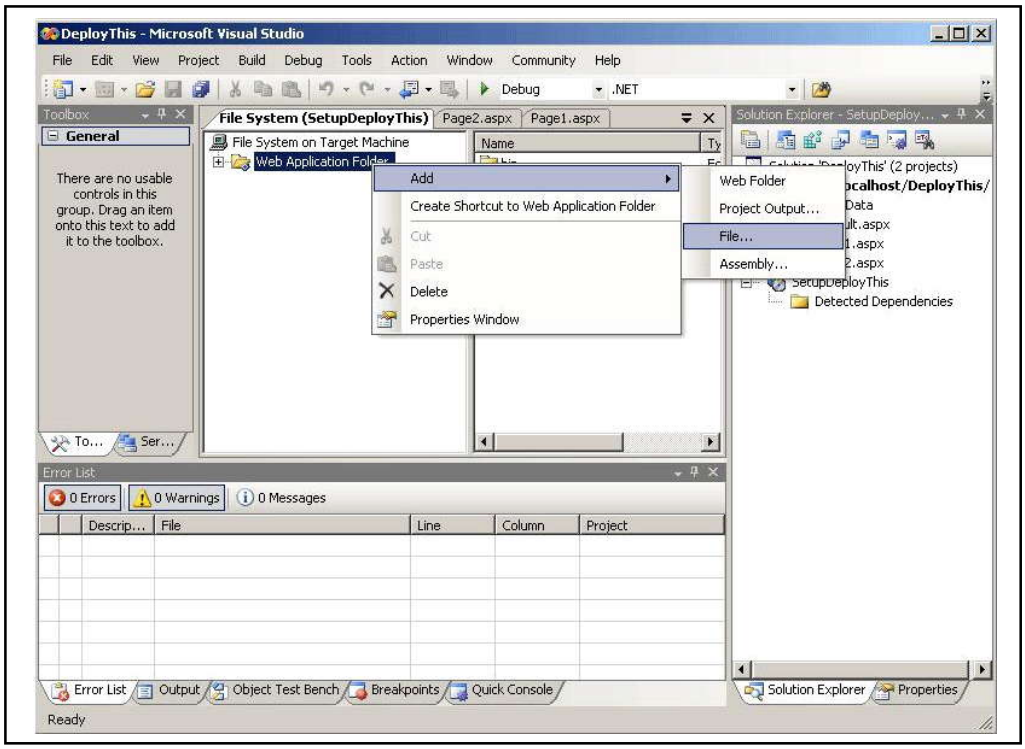


6. Visual Studio will generate a new setup project for you. You should see a screen like this after Visual Studio is done churning:

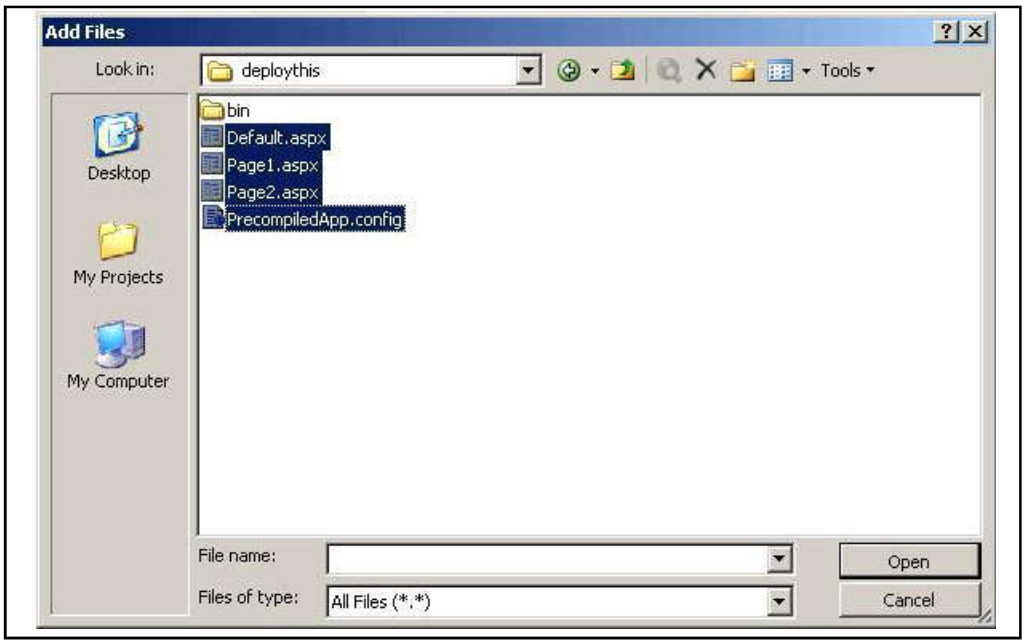


7. Right-click on the Web Application Folder to add the Web files. Navigate to the target directory containing the site code. This will be the pre-compile directory.

Notes

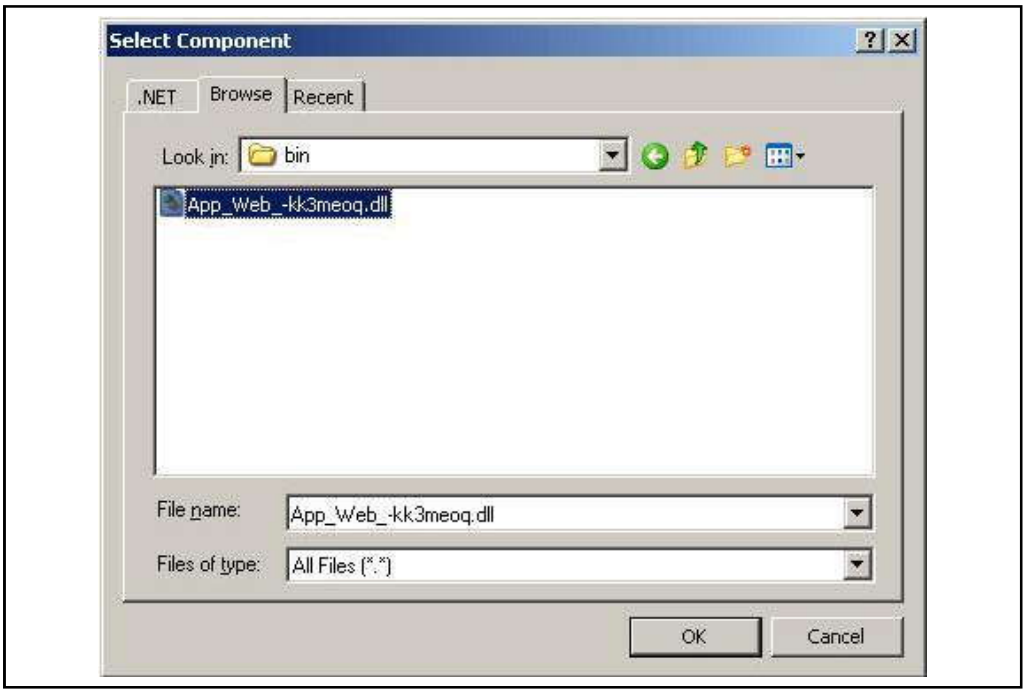


8. Add the Web files from the pre-compile directory.

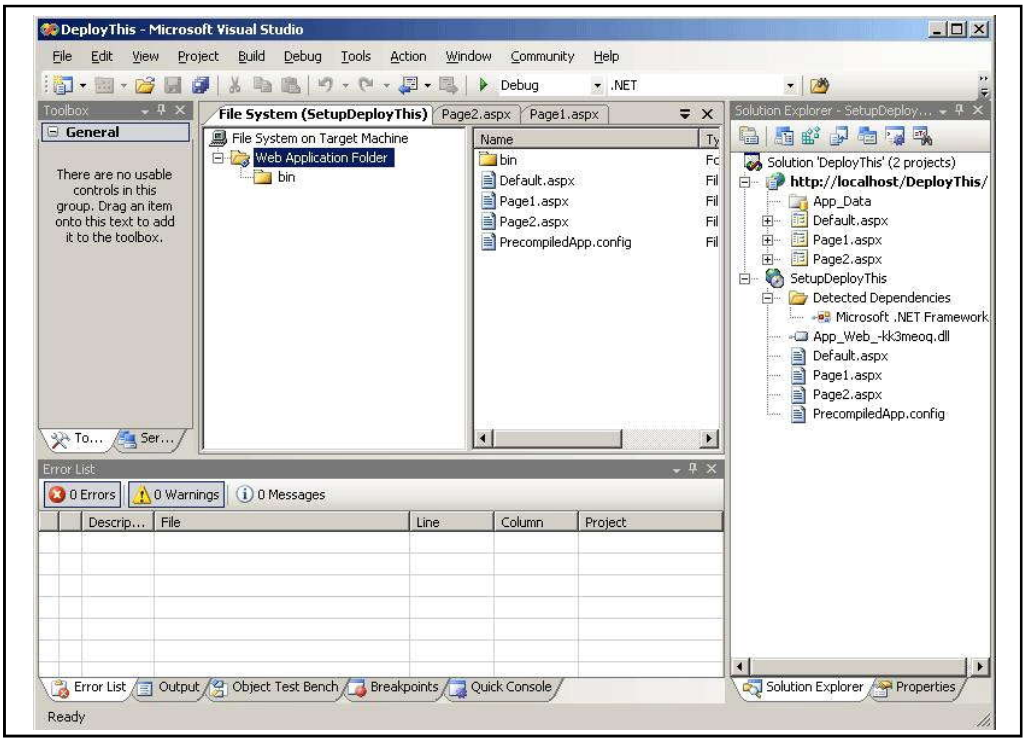


9. Add the dlls to the bin directory by right-clicking on the bin node to get the File Open dialog box. Then search for assemblies in the target directory's \bin directory.

Notes

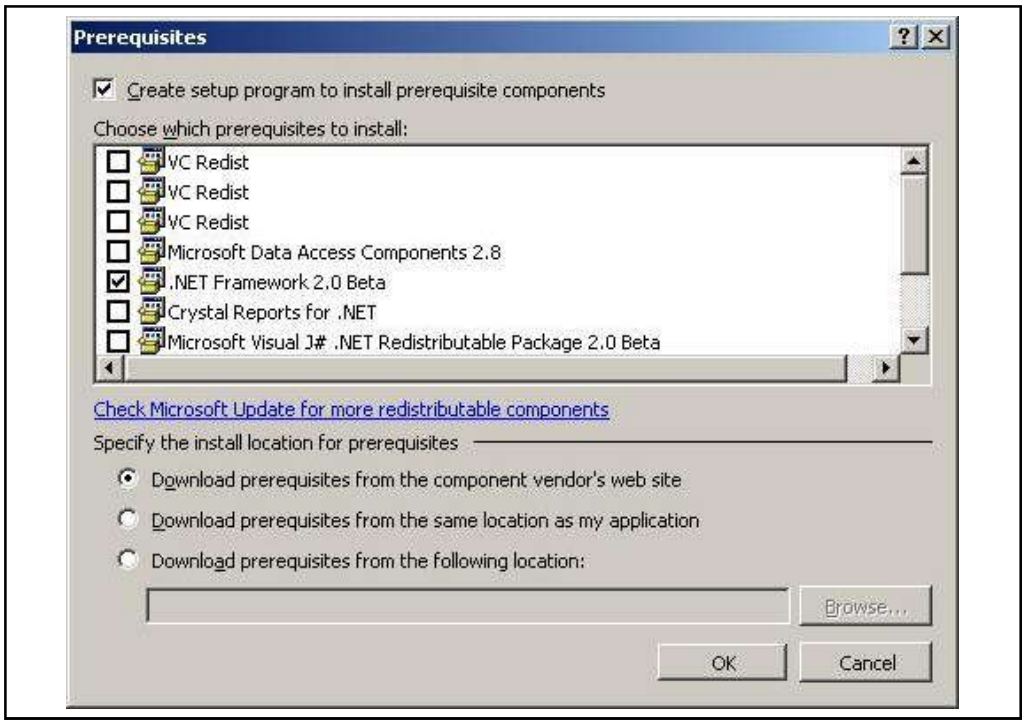


10. After adding all the files, the directory structure should look like this. The bin directory will have the site DLLs:



11. The Setup project properties include a prerequisite dialog box that you may review to ensure that certain prerequisites are installed on the end computer. The following graphic illustrates the prerequisites dialog box. Notice that the .NET Framework box is checked.

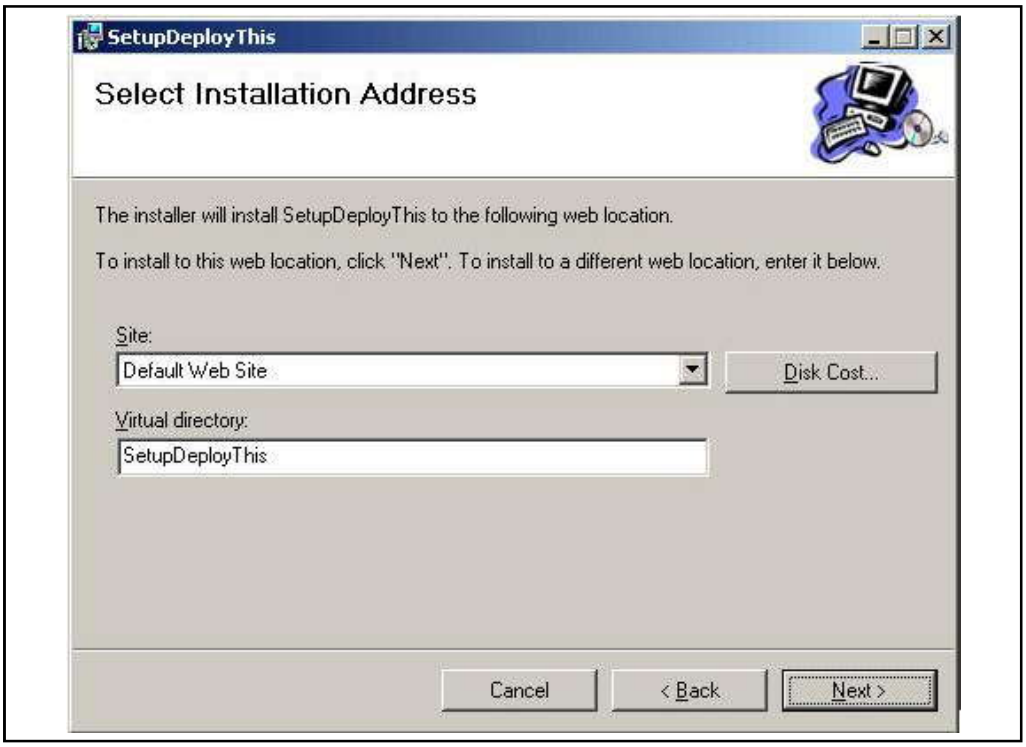
Notes

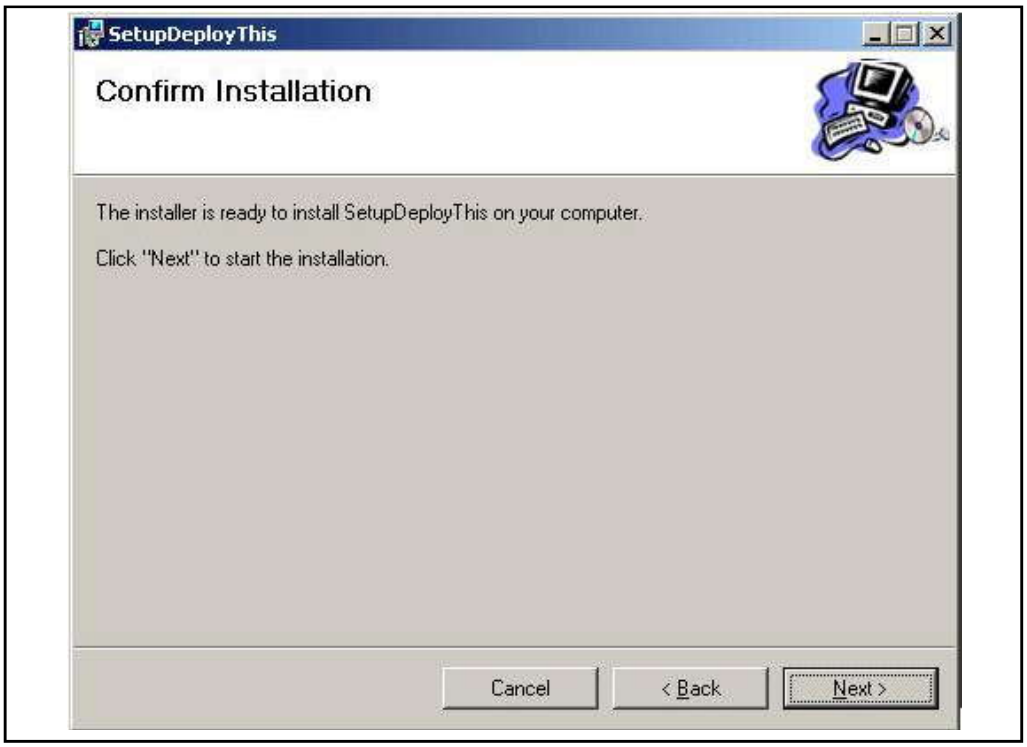


12. Right-click on the SetupDeployThis project and select Build. The resulting MSI file goes in the debug directory of the project.

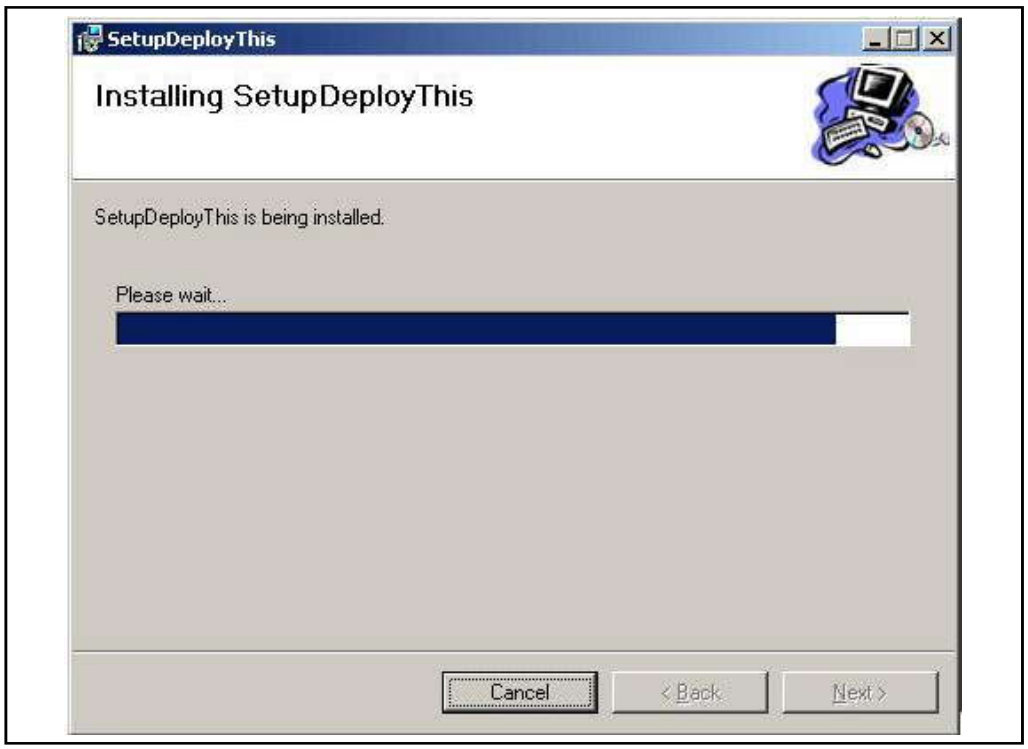
13. Try running the Microsoft Installer file (the MSI file). The MSI file will guide you through several steps as it installs the Web site, as shown in the following graphics:

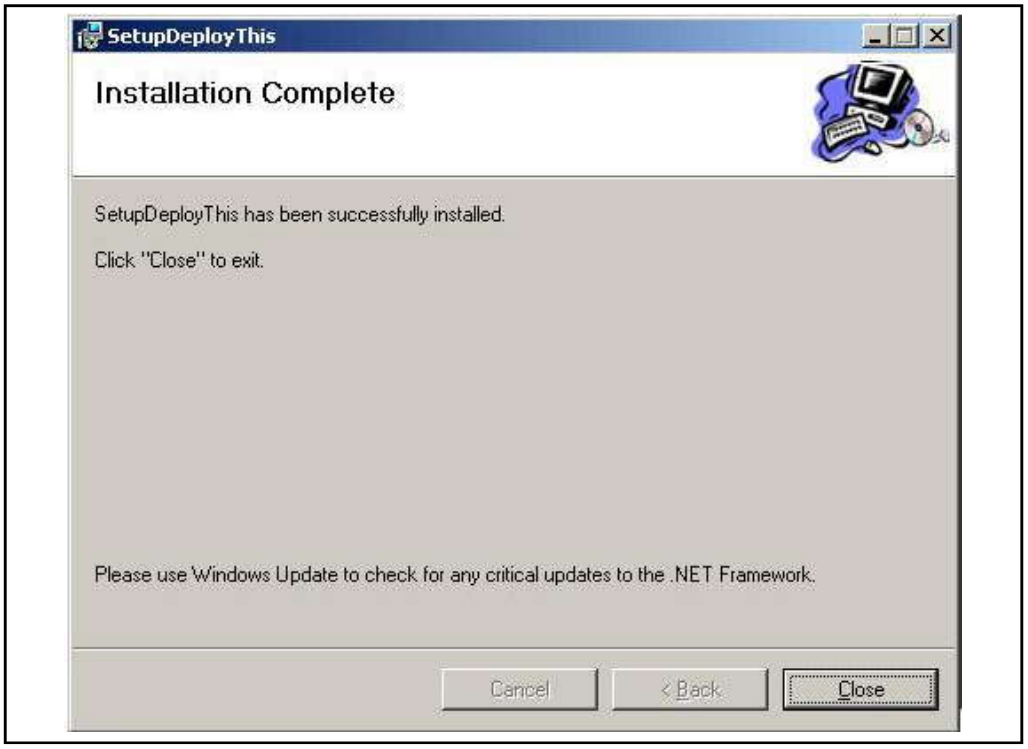
Notes


SetupDeployThis
Select Installation Address
The installer will install SetupDeployThis to the following web location.
To install to this web location, click "Next". To install to a different web location, enter it below.
Site:
Default Web Site
Disk Cost...
Virtual directory:
SetupDeployThis
Cancel
< Back
Next >


SetupDeployThis
Confirm Installation
The installer is ready to install SetupDeployThis on your computer.
Click "Next" to start the installation.
Cancel
< Back
Next >

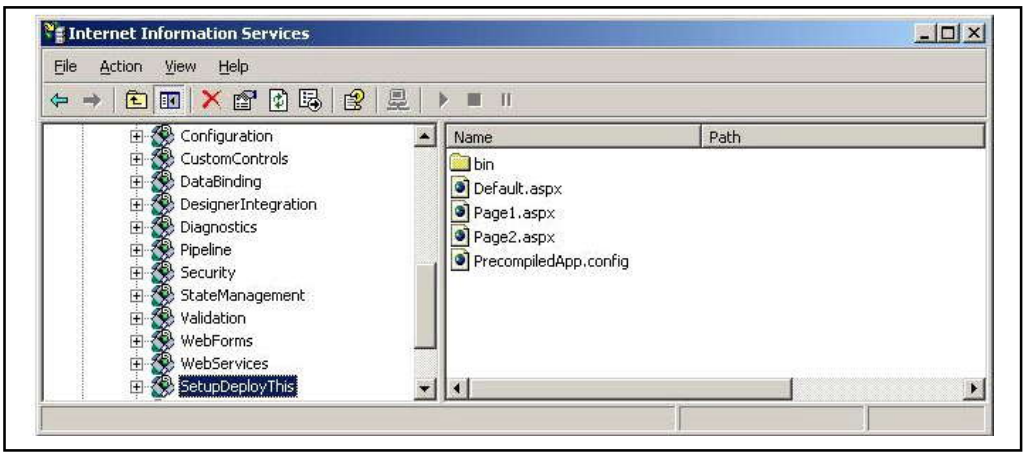
Notes



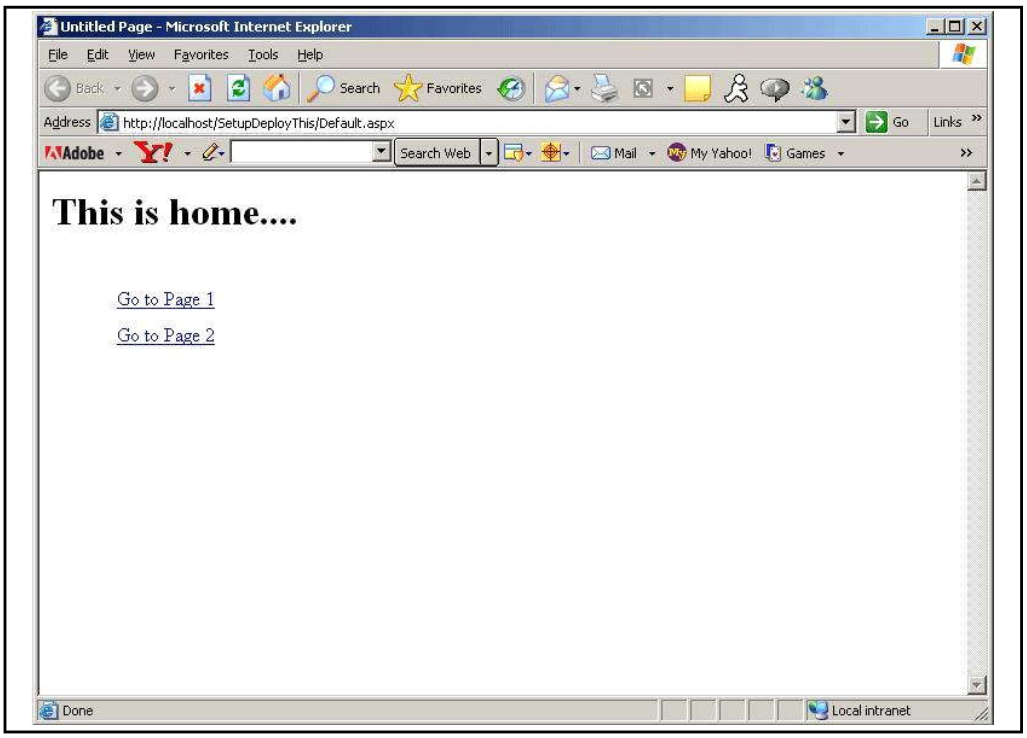


14. Now examine IIS. Refresh the Default Web Site node and look for the SetupDeployThis virtual directory (unless you named it something else during the install process). IIS will have the SetupDeployThis site:

Notes



15. After the site is installed, you can surf to it as you can any other site.



Setting up Install packages is a good way to distribute a Web application across the set of servers. You can push the MSI file out to the server as necessary and run it. However, using an Install package isn't the only way to distribute the application. You may also literally copy the entire directory from a development machine to the server (XCOPY deployment), or you may use some other file transfer mechanism to move the bits.

Task

Discuss the basic purpose of website deployment. How it useful as a programmer or user?

Notes

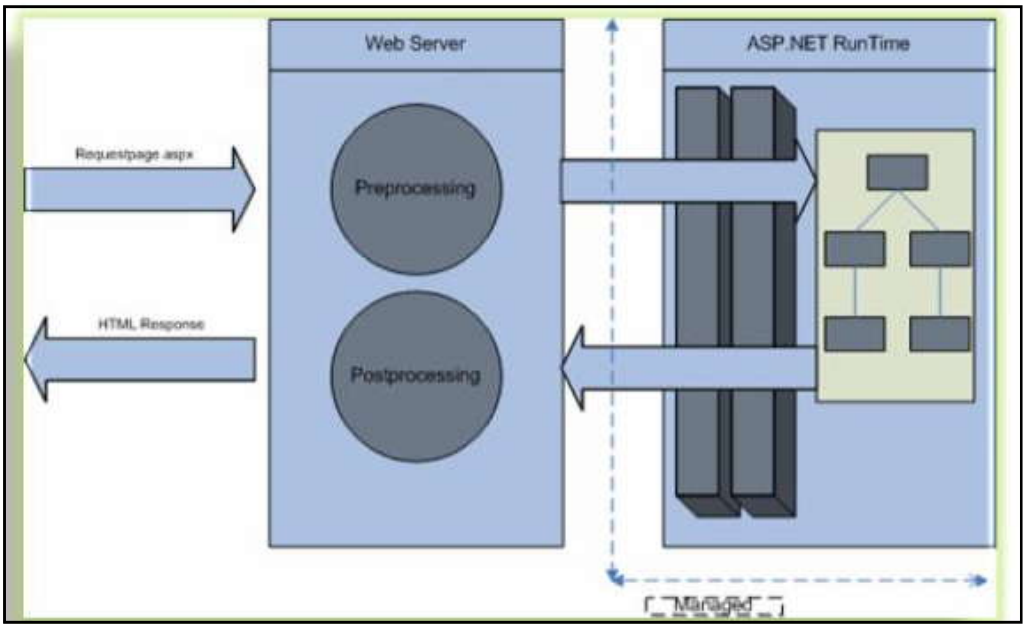
14.3 Deployment of a Website on IIS

For deployment of a website on IIS, first we need to understand IIS, it's request processing and then we'll go for deployment of the website and various configuration available in IIS, their key features, advantages and disadvantages etc.

1. IIS
2. Managing a Virtual directory
3. Managing a Application Pool
4. Deploying application

14.3.1 IIS

IIS at root level, a Windows Service that is responsible for processing requests received on specific ports. For it, a service called the World Wide Web Publishing Service runs on the system. For deploying an application, we cerate a Virtual directory and the convert it into an application. Actually Virtual Directory is nothing more than a configuration entry in IIS for sharing a physical path for access through Web Server.



The complete configuration (with all its settings) is stored in a file, called a metabase, on local system. IIS metabase is a XML -based data store in IIS6.x and it can be configured through IIS management console.

IIS and URL Processing

Normally ,Web application are accessed and processed through browser. Browser sends the request to the Server (Here it is IIS) which actually process the request. The following are the steps that are done by IIS

1. First, IIS examine the requested URL. It also check the port if it is not configured on default port 80, then url also requires port no.

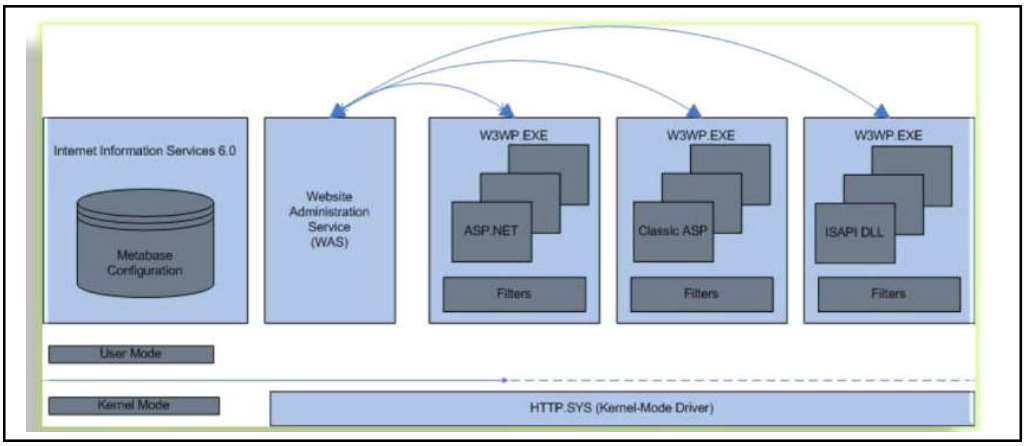
Notes

2. If we have a URL "http://webserver/store/books.aspx" then here web server shows the name of Virtual directory and books.aspx is the requested file.
3. Every file extension is registered in IIS and connected with ISAPI extension and every file is connected to a DLL file.
4. Every file extension is connected to ASP.NET runtime are connected with appropriate aspnet_isapi.dll ISAPI extension. these are automatically added at the time of installation. Using this different version of framework are managed (as, ASP.NET 1.0 1.1 and 2.0) and can be configured for every website.
5. IIS 6 is split in several components: In it a Kernel driver as the picture below is responsible for receiving HTTP requests from the clients then it forwards requests to any process that registers itself for specific URLs. i e any application that registers with the kernel mode driver can receive HTTP requests without running the whole server.
6. IIS launches worker processes which provide mechanism of Isolation and each worker process runs one or more applications, either ASP.NET based or any other type

IIS 6.0 Process Model

The web server is now split into several components. IIS 6.0 includes a kernel mode driver called HTTP.SYS, which is responsible for receiving HTTP requests from clients. And the kernel forward the requests to any process that registers itself to specific URLs.

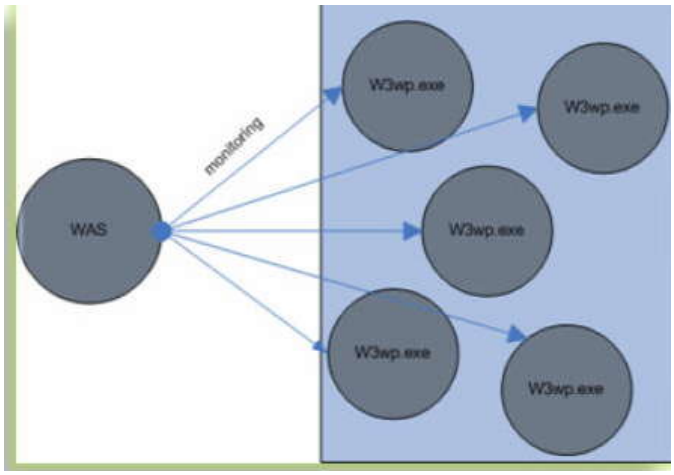
All the reliability and security options are configured at the applications pool level. Therefore, when running ASP.NET on IIS6.0,the classic ASP.NET process model with the configuration of the <processModel> element in the machine.config is desabled ,cause all the options introduced for the <processModel> are configured for IIS6.0 worker process.



WAS monitors each workers process and if one fails it restarts it so that it application doesn't go down abruptly.

1. We can define separate identity for each worker process ie it allows to configure additional isolation through permissions of the account that's configured for the worker process and these are configured through application pools in IIS management console i e every application pool has itself own worker process and every Virtual Directory can be assigned to these application pools. Each application pool can run as many applications as needed.
2. Application pools allows us to easily configure different web applications to run under different account with different resource usage limits and provide more web application isolation

Notes

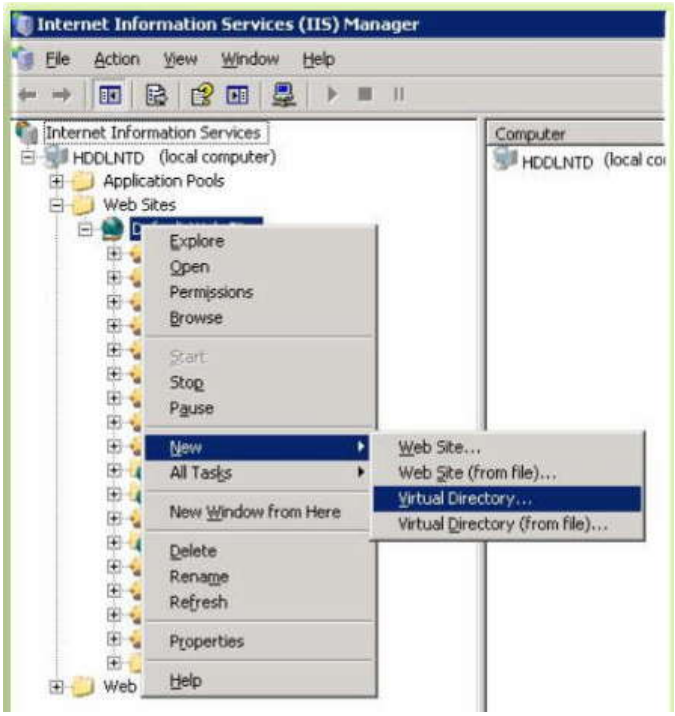


14.3.2 Managing a Virtual Directory

Managing a virtual directory includes creating a virtual directory and its all settings. When IIS is installed a directory named "c:/Inetpub/wwwroot" is created automatically.Any files in this directory will appear as they're in the root of the web server.using the wwwroot directory fro creating a virtual directory makes a very poor organisation so try to avoid it.That's why I coosen the second option as below

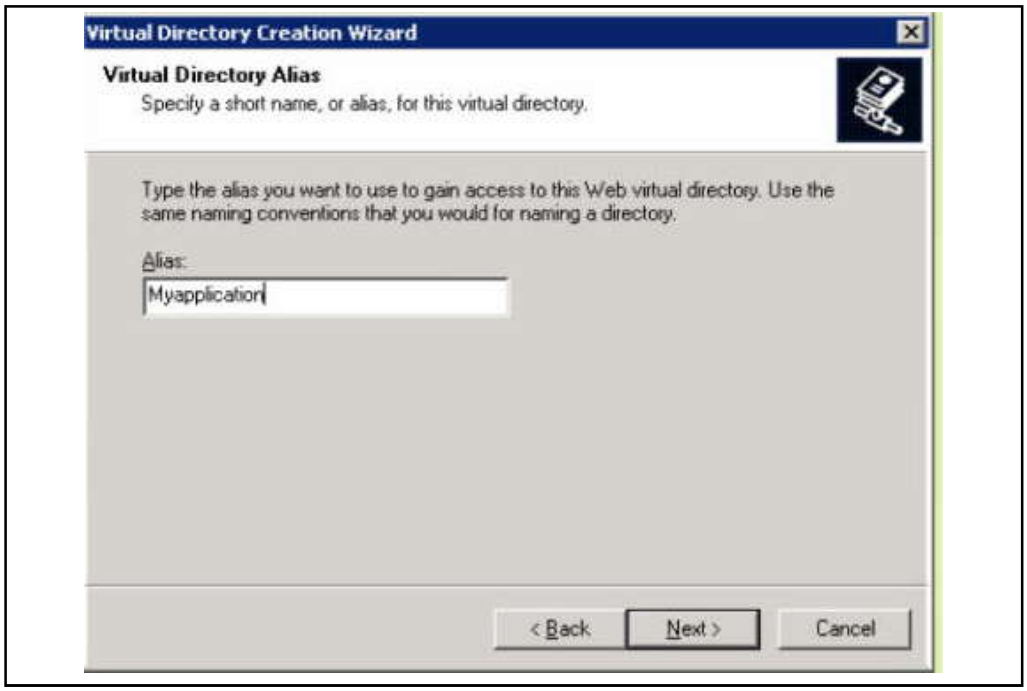
Making a Virtual Directory

1. Go to run and type inetmgr and press enter. You will get IIS management console as below and do as the screen shot.

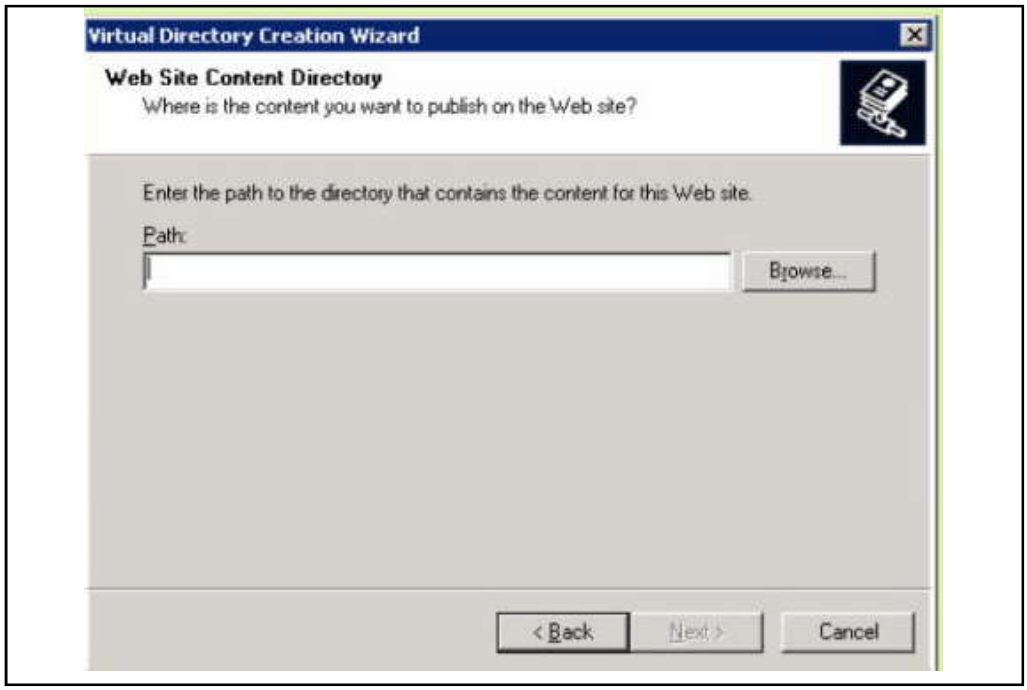


Notes

2. Click next.Set alias of your website as you want

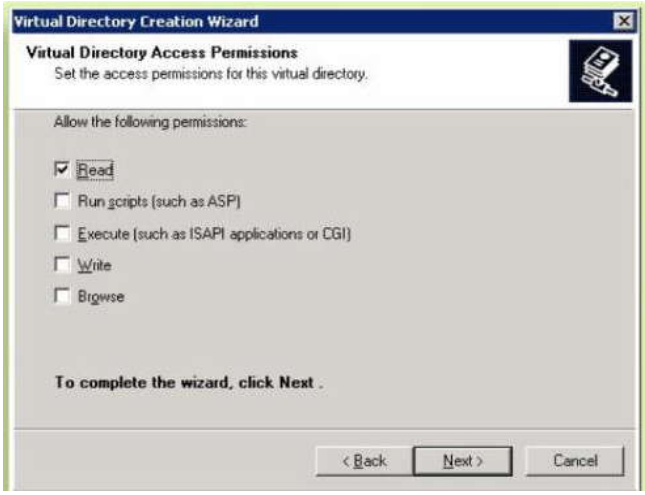


3. Click Next and browse the physical path of your application.



Notes

4. Set Permissions as we have five options as



(a) *Read:* It is most basic and is mandatory to access the webpage of your application.

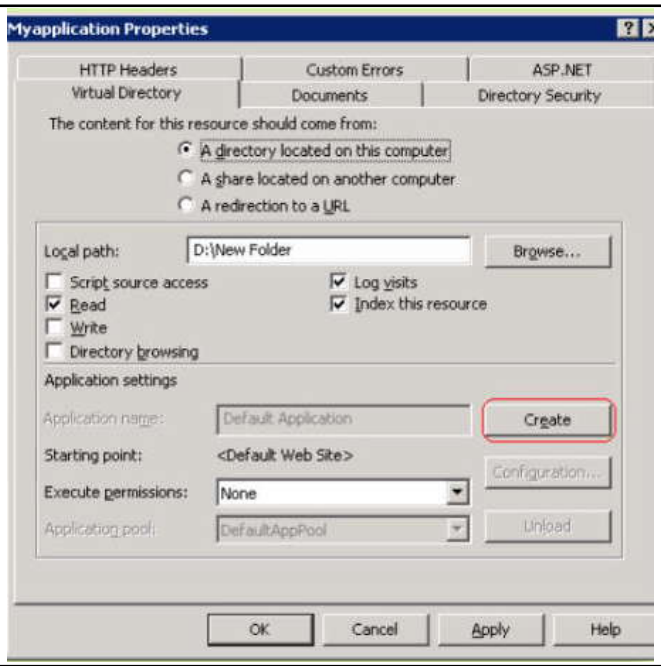
(b) *Run Scripts:* It is required for the aspx pages not for the static HTML pages because aspx pages need more permission sp that they could conceivably perform operations.

(c) *Execute:* This allows the user to run an ordinary executable file or CGI application. This can be a security risk so allow when it is really needed.

(d) *Write:* It allows to add, modify or remove files from the web server. This should never be allowed.

(e) *Browse:* This allows one to retrieve a full list of files in the virtual directory even if the contents of the file are restricted. It is generally disabled.

5. Right Click on the Virtual Directory and go to properties and clock on create to set it as application.

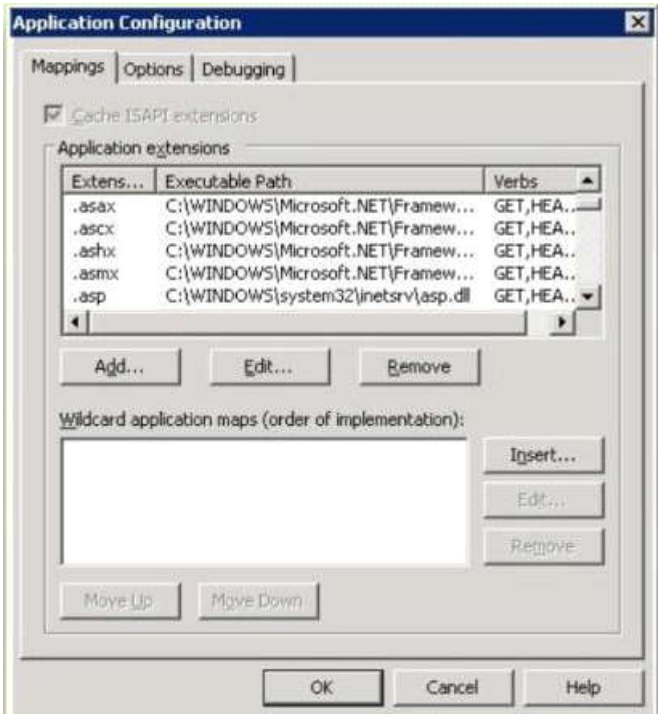


Virtual Directories Settings

Notes

IIS made very easy to configure Virtual directory settings after creation of it. Just Right click on virtual directory and choose properties. And the property window will appear. Below I am trying to explore some most important settings.

File Mappings: As earlier, IIS forwards requests for aspx pages to the ASP ISAPI extension and sends requests for ASP.NET pages to the ASP.NET ISAPI extensions. And IIS decides the ISAPI extension based on the filename extension of the requested URL and these mappings can be done per virtual directory basis. To view it, click Configuration button on the Virtual Directory tab in the properties of a Virtual Directory.

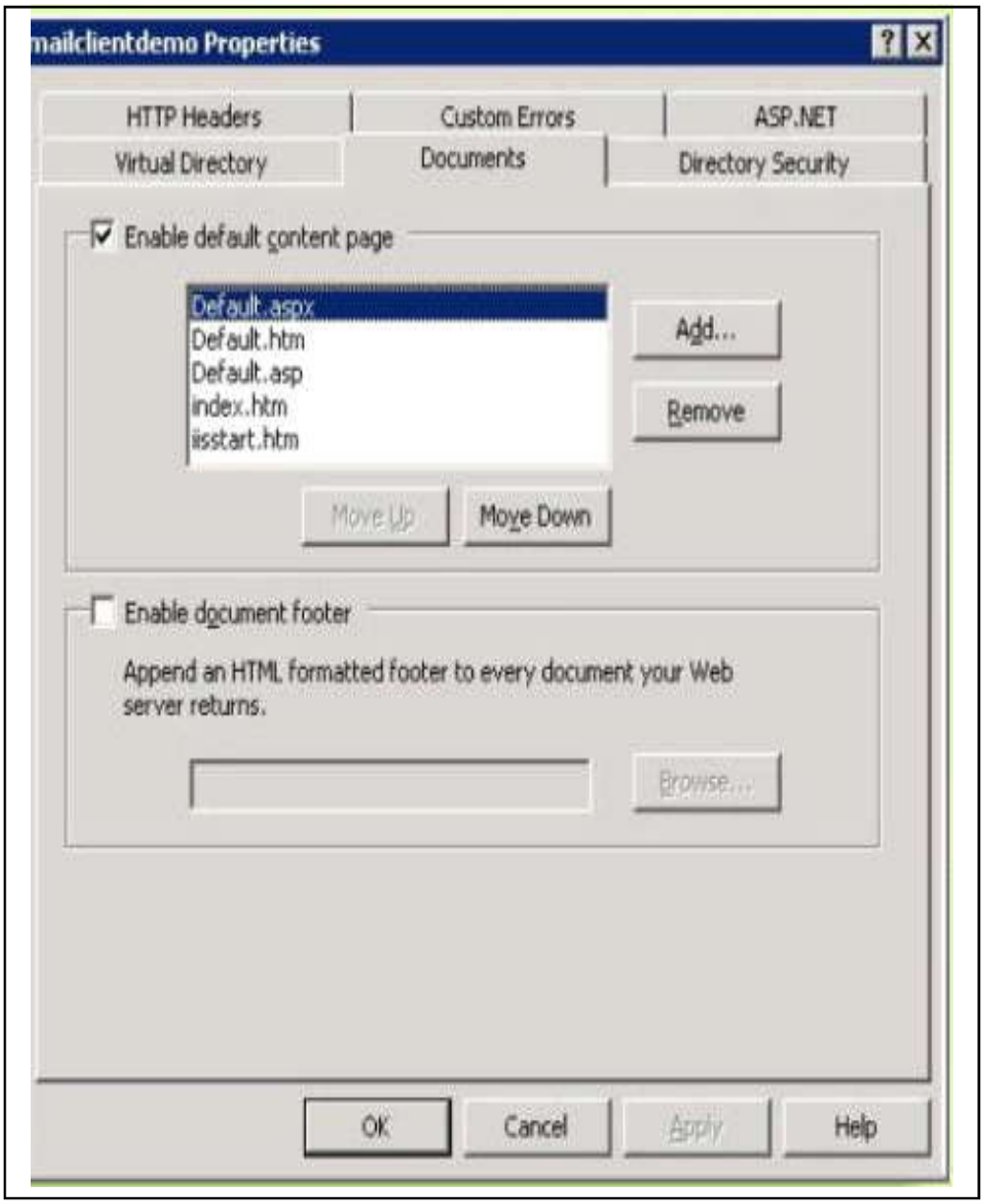


S No	File Extensions	Description
1	.aspx	These are ASP.NET WebPages
2	.ascx	These are ASP.NET user controls. It can't be accessed directly, to access it it must be hosted on ASP.NET pages.
3	.asmx	These are ASP.NET web services, which is used for exposing functionality to other applications over HTTP.
4	.asax	These are for Global application file which is used for global events like when user starts the application.
5	.ashx	These are HTTPHandlers,which allow to process requests without using the full-fledged ASP.NET web-page model.
6	.axd	These are used for the trace.axd application extension, which allows to trace messages while debugging
7	.rem and .soap	These extn identify that IIS is hosting an object that can be called by .NET Remoting
8	.cs,.csproj,.licx,.config,.resx,.webinfo	These files are used by ASP.NET, but that can't be directly called by clientsAlthow,ASP.NET registers them so that it can explicitly prevent users from accessing these files, regardless of the IIS security settings.

Notes

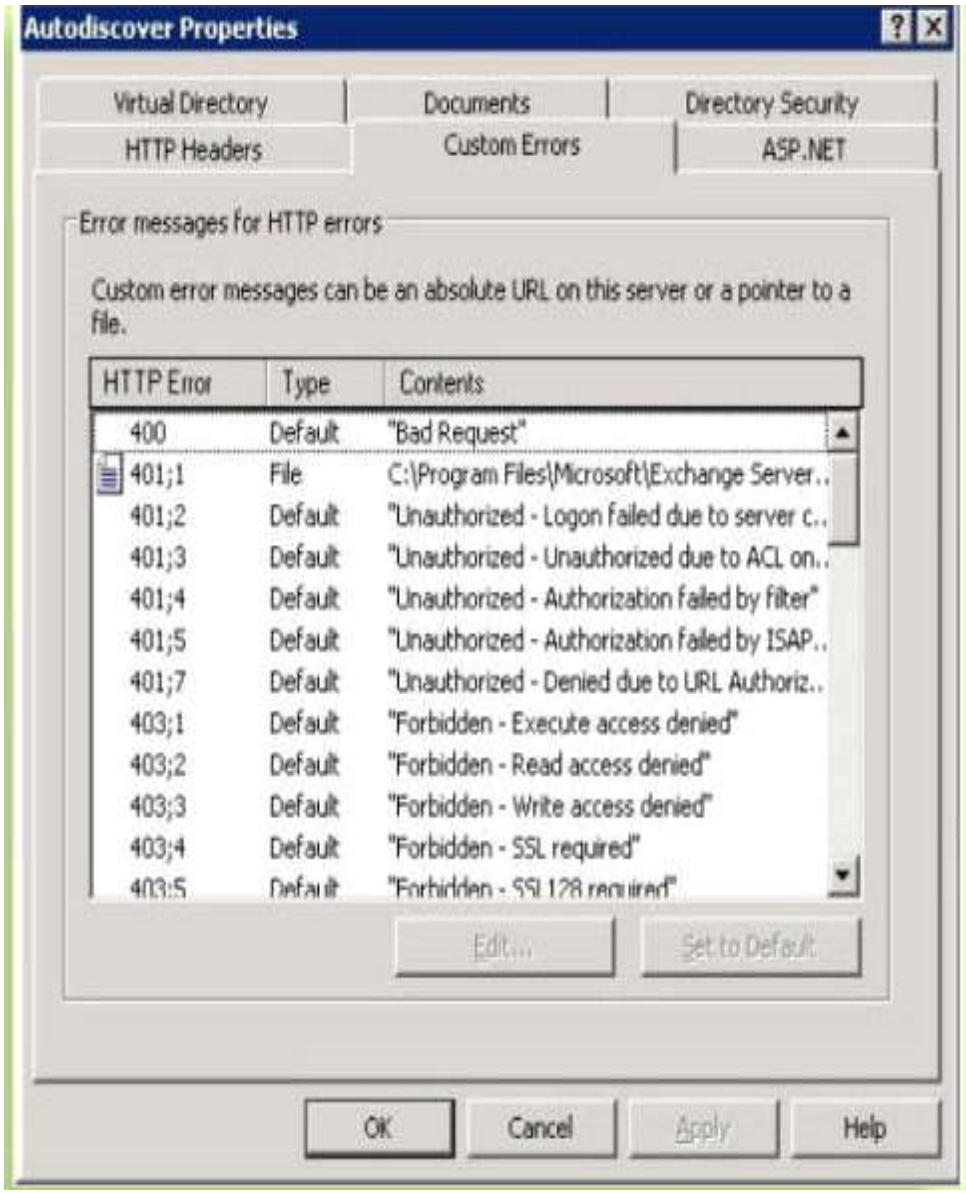
Normally, if we have multiple versions of ASP.NET installed at one point of time, one may want to configure the mappings differently in different directories.

Documents: This tab allows one to specify the default documents for a Virtual Directory. Like if user just type "http://MyServer/MyApplication" then IIS simply redirect to the user to that default page. If none page is found, IIS will return the HTTP 404 (page not found error). To see and set the default page, in the properties of the Virtual Directories, Click on the documents tab



Custom Errors: This tab allows us to specify an error page that'll be displayed for specific type of HTTP errors (see below picture) .One can use ASP.NET configuration to replace HTTP errors or application errors with custom messages. This only works if the web request makes it to ASP.NET service.

Notes



14.3.3 Managing a Application Pool (Worker Processes)

As a default installation, one application pool called DefaultAppPool is created. This runs as a Network Service, every web application runs on default website uses this default pool.

Creating Application Pools

First let us try to know, when we need different application pool as we have already an application pool as DefaultAppPool .Mainly there are three reasons

1. ***Stability problems:*** If on a server there are multiple application are running, If all are on same pool then if this pool have any problem the all the application using this pool are going to affected.

Notes

2. ***Memory Leaks:*** If there is resource intensive application an old application running on a application pool with a memory leak is perfect candidate for regular recycling. In this case, applications running on different application pool, will not get affected.

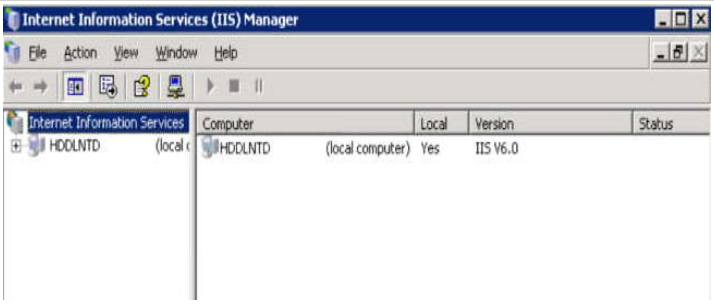
3. ***Security:*** Security configuration is another main reason for having multiple application pool. Let us, we want an application that will be able to write some logs on the client computer, then we should have an application pool which should run on LocalSystem's account other might be doesn't need the same so they can on different accounts.

4. ***Administration:*** This is also one of the reasons for having separate pools. Might be on a server, there are multiple sites hosted for separate parties then it doesn't allow to access the resources of another pools.

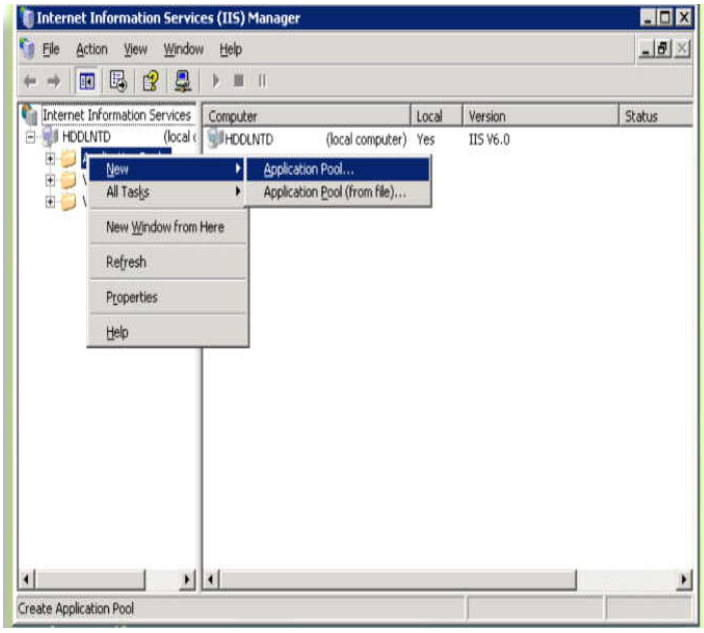
Notes Recycling an application pool (worker process) means stopping the old worker processes which have already take lots of resources and start new instance of it for the application pool.

To create the application pool follow the below steps (Please keep in mind that IIS should be installed on the machine)

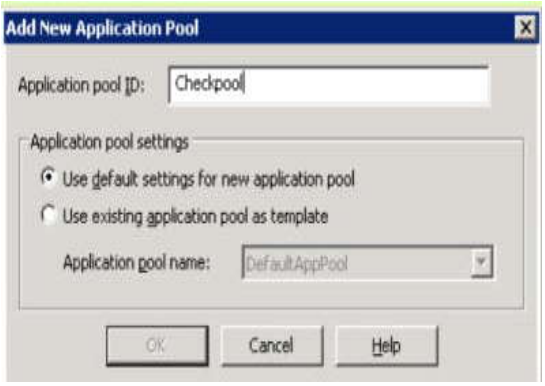
1. Go to run -> type inetmgr then click ok. You will get the following window.



2.



3. **Notes**

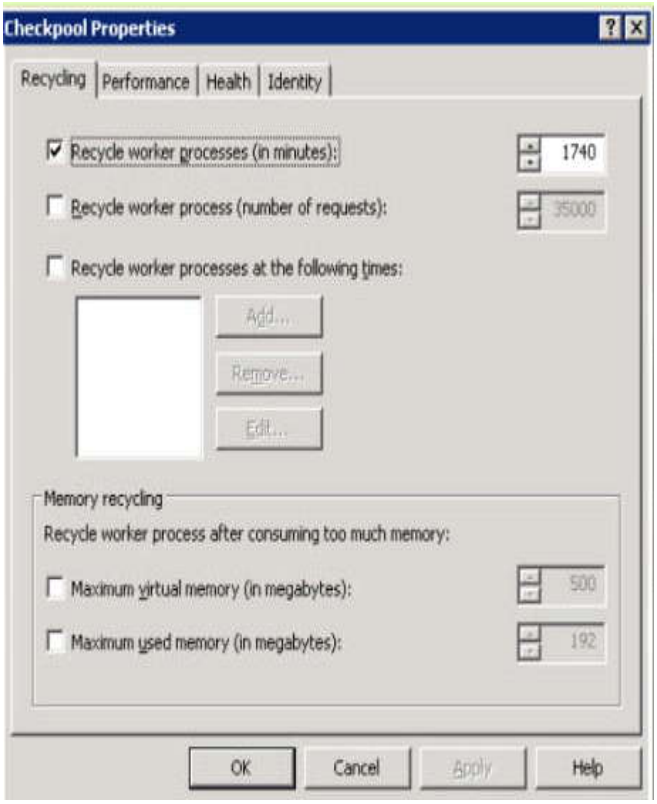


Here you have two options

(a) *First*, if you want to use default settings (use it when you want to create it with new settings)

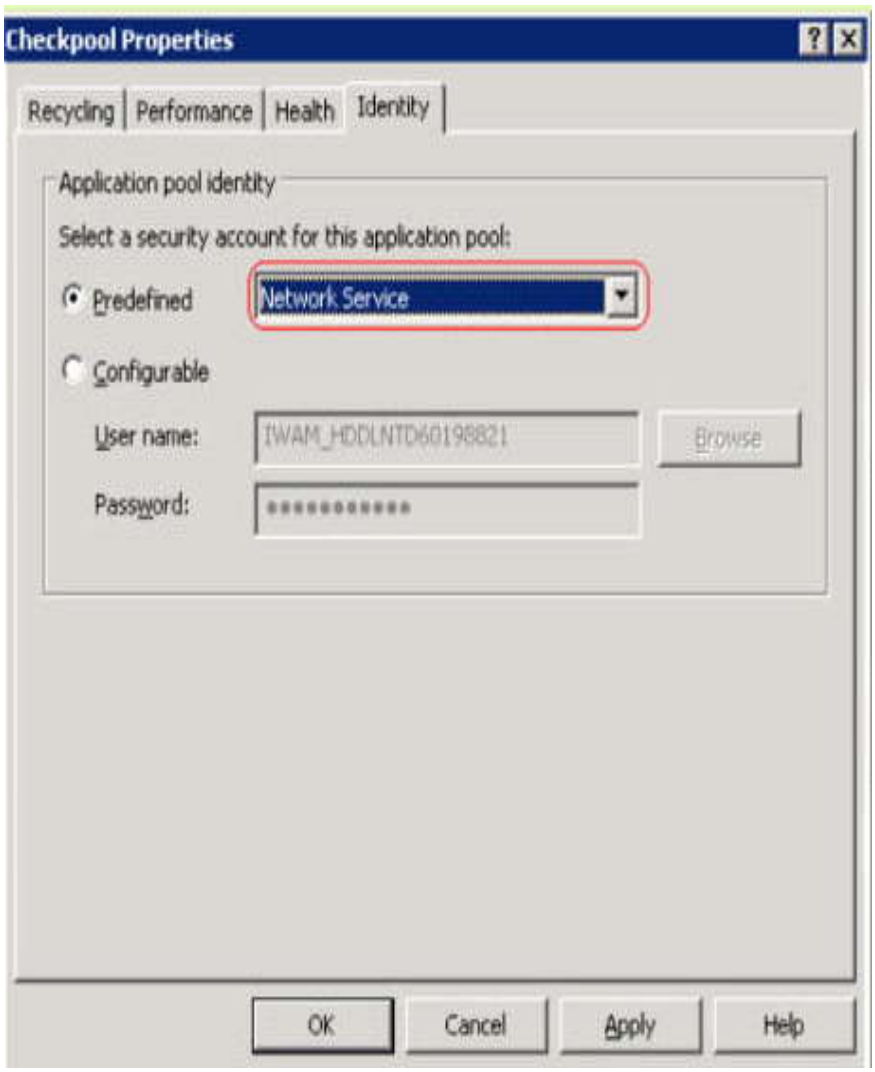
(b) *Second*, If you want to have a new pool with just same settings as some another pool then you can select this one and change the template pool from the dropdown (It'll save your time from doing the same settings again)

4. You can configure the application pool by right click on the pool created and select properties



5. You can configure the identity of every application pool by selecting Identity tab from the property window of application pool as above window

Notes



Here in the dropdown, one has three options:

(a) *Network Service:* This account is restricted account with least privileges in the three. This is mainly used for applications that require access to network and need to be accessed from other machines in the network.

(b) *Local Service:* This account is having more restricted than Network service and mainly to be used for services that don't require additional network access.

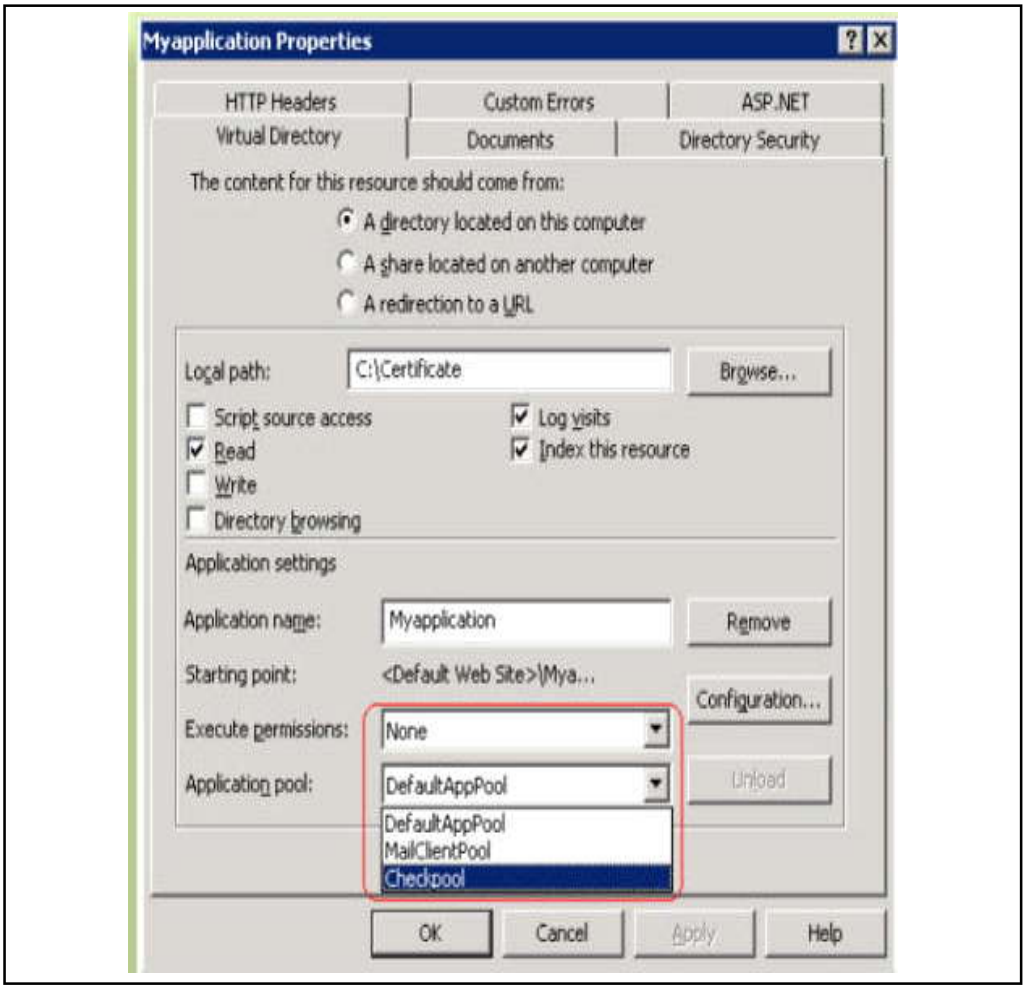
(c) *Local system:* Generally this account is not recommended to use, because it is most powerful account of the system. It can perform any action on the local system. so the basic motto , one should provide that much of privilege that the application needed so it can't be hacked easily.

You can also configure it as selecting your own user account by specifying the Windows user name and password for this account.

Notes

Application Pools Configuration

Right Click on the application and go to Virtual directory tab change the application pool and click on apply button.



14.3.4 Deploying Application

Deploying is nothing more than copying the published code of the application to target machine and configure the environment as needed. this is true for simple application but for application using database or access other resources, one need to perform additional steps.

1. Copy required file to target machine: Check whether all assemblies are in place. If not use install it using gacutil.exe.
2. Create and configure the Database
3. Add IIS file mappings as per requirement
4. Update web.config for any application settings or connection strings etc.
5. Rest settings can be done as per requirement as we discussed above like application pool etc.

Notes

Task Specify the uses of virtual directories.

Case Study

Web Services Deployment

Web services are "in production." While much of the trade press insists on covering Web services as a future technology, it is solving problems for organizations today. This case study profiles how one successful public deployment looks to the programmers who coded it.

Are Web serivces projects manageable? This is a crucial uncertainty that plagues many deployment plans. Readers of the developerWorks Web services zone already recognize that Web services is a proven technology that will be of enormous importance in the near future. What are the consequences, though, for project planning and fulfillment? How does Web services work differ from earlier generations of client-server or host-based programming? Lucin's public Short Message Service (SMS) sheds light on these questions.

Categories of Services Use

First, consider the industrial context for Web services. Most such projects today fall into one of four categories:

1. Content management, collaboration, and syndication. Public leaders in this have been Dave Winer's Userland, O'Reilly and Associates' Meerkat facilities, and Jon Udell's development and articles for *Linux Magazine* and Byte.com.
2. Internal service consolidation. Not unlike the explosion of "intranets" in 1995 and 1996, quite a bit of Web services development is hidden from public view behind firewalls. Organizations strategically dependent on a variety of technologies are consolidating their internal interfaces as services.
3. CORBA and DCOM substitution. Projects might be feasible with CORBA (IIOP, more properly) or DCOM, and would have been executed that way last year. This year, though, decision-makers have begun to choose Web services as a technical substitute for these competing distributed-computing protocols.
4. "Programmer-centered" projects that exploit Web services' public nature and interoperability in a way CORBA and DCOM don't effectively rival.

This case study falls in the last of these categories: Web services brings crucial benefits for this application that CORBA and DCOM can't match. DCOM essentially limits clients to those running on Windows desktops. On the other hand, CORBA is "heavier" than needed for the simple processing the gateway requires. Moreover, real-world CORBA projects have the reputation of stumbling over firewall and interoperability problems that Web services handles gracefully.

Development Team: Vital Statistics

Lucin PLC is a small software consultancy headquarterd in Newport, Wales, in the United Kingdom. Mike Clark is its founder and managing director. It has specialized in transaction-oriented client-server applications for over a decade. In February 2001, three engineers

Contd...

Notes

began work on a small demonstration of a Web services-to-SMS gateway. A few weeks later, in March, it opened the service to the public.

"Short Messaging Service" or "Short Message Sending" is widely supported in wireless telephones in most European and Asia-Pacific countries, although rarely in the USA. Wildly popular in Japan and elsewhere, SMS allows telephone users to compose short textual messages using the telephone handset, and transmit them *asynchronously*. Think of it as instant messaging for cellphones.

It's natural to bind together the pertinent telephony and computing protocols so that computers can originate and perhaps receive such messages. That's Lucin's achievement in its Web services-to-SMS gateway; it is the first of several such for-fee gateways it plans for the months ahead:

1. Web service to pager
2. Web service to fax
3. Web service to speech
4. Web Service to TAPI (telephony application programming interface)

In Lucin's plan, it'll be only a short time, probably the final quarter of 2001, before it has the capability to dial a conventional telephone, and read out an arbitrary message in a synthesized voice.

The starting point, though, is the integration of Web services and SMS. Let's look at that first step in more detail.

Construction of a Web Service

The SMS gateway is a small software project. Much of the media publicity around Web services envisions integrating enterprise applications on large supply chain projects. Clark insists that this misses the point of the "instant gratification services" already moving into production for modest, well-specified projects. He deliberately chose SMS as a model. As Clark indicates, "[SMS] fits this [model] perfectly as a Web service that's great to use," but can afford occasional outages or downtime.

Current infrastructure enforces this kind of tolerance. Networks simply are not 100% reliable or ubiquitous yet.

Some Web services vendors are hawking "revolution" with the intelligence in their Web services tools. However well this software works in the laboratory, it appears few organizations have enough well-connected customers to layer so much weight over existing networks. It doesn't matter how clever the error-correction and transactional algorithms one programs: until basic network connectivity gains an order of magnitude in price and reliability, there's little point to building elaborate Web services. Clark summarizes, "Don't expect your users to be happy with response times. It's currently out of our hands; maybe in two years time it will change."

The inadequacy of the current networking infrastructure impels Clark to manage Lucin conservatively: "We've deliberately budgeted for a three year cycle, meaning we can pretty much last three years with no significant customer base. This is because my belief is that it's going to be a long trek before Web services are thought as reliable and trustworthy enough for businesses to use in the long term."

Contd...

Notes

In the meantime, the kinds of service that do satisfy users might be even simpler than you expect. Web services developers think of the typical high-level design as one that pulls in data or objects across the network, performs calculations on them, then uses Web services protocols to deliver results back across the network. Lucin sells a number of Web services products that fit this model and has several internal tools to support such development.

For the SMS gateway, though, Lucin eventually settled on an even more primitive approach. Clark explains that ". . . if you know the interface then you simply need to construct a simple string variable and substitute the values that change at the time you send the data to the remote Web service. This saves time and the overhead of having to reference [a SOAP toolkit or module]."

Lucin does this simple string substitution, and the other light-duty calculations involved in the production version of the SMS gateway, with Visual Basic, hosted on an NT Server 4.0. This supplies more than enough performance for the several hundred messages transacted each day (most often between 300 and 800).

The gateway's simplicity extends to its authentication: a user-password combination sent in plain text by way of HyperText Transport Protocol (HTTP). Clark contends, "You would not expect to use HTTPS [secure HTTP] for signing onto a Web site, so why use HTTPS to get users to pass down username and passwords so that you can validate them for using your web service." Pressed about the frequency of HTTPS use, Clark criticized its run-time performance, and concluded, "we may be using HTTPS ability at some stage, but it will almost certainly be by customer request."

"One of the beauties of Web service development," according to Clark, "is the flexibility that comes from use of simple, robust parts." The gateway connects on the back end to a metered SMS interface provided by a telecommunications vendor. Twice already, Lucin has switched providers for this back end, but without any interruption in service to its customers. Users of the gateway just receive acknowledgment of receipt of their requests; they don't have to know or care about the back end.

14.4 Summary

- In this unit, we looked at how the various Visual Studio projects affect the end deployment strategy for your Web site. Visual Studio provides several models, including
- HTTP sites that use IIS on the development machine
- File system sites that exist in the development file system, using the Web server built into Visual Studio
- FTP sites, where the bits are transferred to the target server via FTP
- In addition to copying the software directly to the deployment machine, you may also pre-compile the application before copying it.
- By pre-compiling, you save the first end user to hit your site the few seconds it takes to compile the site. Of course, the subsequent hits take a much shorter time. However, if you foresee the site churning a lot, it may be worthwhile to pre-compile for performance. In addition, you may pre-compile the application so as to deploy it using an installer or a copying technique.

Notes

14.5 Keywords

API (Application Program Interface): A set of programs, code libraries, or interfaces used by developers to interact with a hardware device, network, operating system, software library, or application.

Deployment Manifest: The part of an application that tells the system how to install and maintain an application.

Deployment: The process of installing an application, service, or content on to one or more computer systems. In .NET, deployment is performed using XCOPY or the Windows Installer. More complex deployment applications, such as System Management Server, can also be used.

FTP: A standard method of transferring files on the Internet. If you control a computer, you can give other users access to specific files on your computer without having to provide an account and password for every possible user.

Managing Virtual Directory: Managing a virtual directory includes creating a virtual directory and its all settings.

14.6 Self Assessment

Fill in the blanks:

1. FTP stands for .....
2. HTTP stands for .....
3. The ..... site option represents a reasonable means of getting files from your development environment to the hosting site.
4. Pre-compiling a Web site involves using ..... tools.
5. Web application are accessed and processed through .....
6. ..... made very easy to configure Virtual directory settings after creation.
7. ..... these are ASP.NET user controls.

State whether the following statements are True or False:

8. The source code for the project is stored in the IIS virtual directory.
9. IIS is not a Windows Service that is responsible for processing requests received on specific ports.
10. IIS 6.0 includes a kernel mode driver called HTTP.SYS.

14.7 Review Questions

1. Describe HTTP projects.
2. What do you mean by file system projects?
3. Describe the purpose of web site deployment.
4. How will you deploy the web site? Explain.
5. Describe how will you copying the web site without using the IDE.
6. How will you create a web site installer? Explain step-by-step.

Notes

7. Write short note on IIS.
8. What do you mean by virtual directory?
9. Explain how will you manage virtual directory.
10. Distinguish between IIS and URL processing.

Answers: Self Assessment

1.	File Transfer Protocols	2.	Hyper Text Transfer Protocols
3.	FTP	4.	command line
5.	browser	6.	IIS
7.	.ascx	8.	True
9.	False	10.	True

14.8 Further Readings

Books

Bill Evjen Willey, *Professional ASP.NET 3.5 in C# and VB*., Publications, 2008.

Bill Evjen, Jason Beres et. al., *Visual Basic.Net Programming Bible*, Wiley India

Evangelos Petroutsos, *Mastering Visual Basic .NET Database Programming*, Asli Bilgin.

Matthew MacDonald, *Beginning ASP.NET 3.5 in VB 2008*, Apress Second Edition.

Paul Dicinson and Fabio Claudio Ferracchiati, *Professional ADO.NET with VB.NET*, a! Press, 2002.

Richard Lienecker, Using ASP.NET, Pearson Education, 2002.

Stephen Walther, ASP.NET 3.5 Unleashed, Pearson Education.

Online links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com