An Introduction to ASP .NET

using Visual Basic .NET

Peter McMahon
peter@dotnet.za.net

(Incomplete Draft Version)

Table of Contents

Chapter 0

Author's Note

This book started as a project for California-based publishers Apress in December 2000. I began writing in January 2001 with the wildly optimistic goal of completing the book by August. Needless to say I didn't even come close. In the following months the market for books targeting new .NET developers became increasingly competitive, and unfortunately Apress were forced to cancel the project.

What you see before you is the state of the book as it was circa April 2002. Some of the chapters have gone through the technical review and copy editing phases, others have not, so the quality within the book varies somewhat, and is far from professional. Some chapters are incomplete, and some don't make an appearance at all. However, I figured I'd might as well compile what I have and release it rather than continue to let the writing rot on my hard drive. Hopefully this e-book draft will be useful to people beginning with .NET, Web Forms development, or both.

Take note: this book was written before the release of .NET v1.1 or v2.0, and uses Visual Studio .NET 2002. Fortunately not much changed between Visual Studio .NET 2002 and Visual Studio .NET 2003, so although all the screenshots in the book might look at little different to what you'll see in the .NET v1.1 release version (and indeed Visual Studio 2005), most of the code and concepts should be very similar, if not the same.

I'd like to thank everyone involved in the project during 2001 and 2002. Gary, Karen, Dan, Nicolle, Grace, Tracy and everyone else at Apress, thanks for your efforts – I learned a lot from you all. And to everyone who read early chapter drafts and gave me feedback, I really appreciate your words of advice and encouragement! In particular I'd like to thank Mark Balasundram, Brad Simon, Jared Blanchard, Tom Pester, David Palmquist and Marc Pienaar.

– Peter McMahon

June 2005

Chapter 1

Introduction

The Internet has grown at an enormous rate over the past few years, and it has almost certainly affected the lives of most people who use computers. In only a few years, the Internet has evolved from a platform for publishing “online brochures” to an entire architecture for developing dynamic, distributed applications. Developing these applications was previously extremely difficult and tedious. The potential was huge, but the tools consisted of nothing more than text editors in which applications would be coded from scratch. This method of writing Web applications required extensive knowledge of the “plumbing” of the Internet, and there were a very limited number of languages to use. These factors made Web application programming available only to dedicated programmers who had the time to invest in learning rudimentary interfaces and unfamiliar programming languages.

Several projects and products now drastically simplify Web application development by providing easy-to-use object models and familiar, commonly used programming languages. Microsoft has been a leader in this field with its Active Server Pages (ASP) technology. With the latest release, ASP.NET, which is an integral part of the .NET Framework, Microsoft has attempted to build a technology that will be very familiar to the millions of Visual Basic programmers. ASP.NET enables VB programmers to easily apply their Windows application programming skills to Web application programming without alienating existing ASP programmers—rather, it makes Web application development drastically simpler for them, too.

Who Should Read This Book

This book is aimed primarily at three groups of people. First, those readers who are Visual Basic programmers and wish to learn how to develop Web applications using ASP.NET by applying what they’ve learned in Visual Basic. = Second, current ASP programmers who wish to learn how to become more productive using the completely new, yet familiar, ASP.NET programming model and the Visual Studio .NET IDE. Third, current ASP and Visual Basic programmers who wish to merge their skills to increase their productivity.

This book assumes no previous knowledge of building Web applications or even simple, static Web pages using hand-coded HTML. There is a chapter dedicated to getting Visual Basic programmers without any HTML or Web building knowledge up to speed. However, this book does assume previous experience with the Visual Basic programming language or a previous subset thereof, VBScript. Included is a section showing the differences between VBScript (and previous versions of Visual Basic) and the Visual Basic .NET language that should prevent some of the subtleties of the language from causing any problems. Knowledge of object-oriented programming is advantageous, although not essential.

How This Book Is Organized

This book is intended not only as a guide and walk-through of ASP.NET fundamentals, but also as an introduction to numerous intermediate and advanced concepts to help you build robust, functional, exciting, and scalable ASP.NET applications. To this end, the book can be divided into two major sections: The first section is from this chapter through Chapter 7, and the second section is from Chapter 8 to the end of the book. The first seven chapters cover the fundamentals of building ASP.NET Web Forms, and you should read them in sequence, as each chapter builds on the knowledge gained from the previous one. The second section of the book deals with a variety of topics, from data access using ADO.NET to XML to caching, that don't follow any particular order—you can read them when you need or want to learn about a particular aspect of ASP.NET. These chapters generally use concepts or techniques that are pertinent to the topic at hand and that have been discussed in the first section of the book.

Chapter 3 is a guide to HTML and other client-side Web programming technologies that VB developers who haven't built Web pages before should definitely read. However, current ASP developers should skip that particular chapter, as they'll probably already know and understand all of the material covered in it. Chapter 3 is an exception, though: There aren't any other chapters in the book targeted specifically toward either previous VB or ASP developers.

What Are Web Forms?

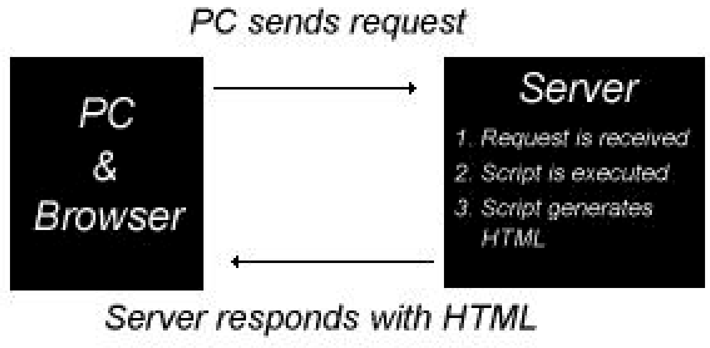
Web Form simply refers to an ASP.NET page. Microsoft coined the term "Web Form" as it accurately describes the process of ASP.NET page design in Visual Studio .NET. As I'll discuss later, Microsoft has made the front end, or user interface, design of Web applications almost identical to the methods used traditionally to design Windows application user interfaces in Visual Basic (which are now referred to as Win Forms applications).

What Is ASP.NET

To fully understand what ASP.NET is, you'll need some background and a brief history of Web application programming. The following sections present a short explanation of server-side programming and a summarized history of server-side programming methods, which should help you greatly in understanding exactly what ASP.NET does and why it's such a groundbreaking technology.

Server-Side Programming

Simply put, *server-side programming* is programming that is done where the code is executed on a server. With regard to Web application server-side programming, this code is executed when a request is received from a visitor to a Web site, and it generates code that Web browsers understand (HTML) and sends it back to the client (the Web browser), as illustrated in Figure 1-1.



Insert 0015f0101scrap.tif FPO

Figure 1-1. Server-side programming model

As you may notice, there's a significant amount of *plumbing,* or code that's always required in order for the application to run. For example, the application must always return HTML code to the client. Because the Web uses the HTTP protocol, the application must understand these HTTP requests and, equally important, respond using correct HTTP syntax. It is for this reason that several technologies have been developed (and have evolved) to make writing Web applications less tedious and more productive by reducing the amount of plumbing that the programmer is required to perform.

Brief History of Server-Side Programming Technologies

Since the Web began in the early 1990s, the original concept of serving Web pages to users has advanced greatly. In the beginning it was only possible to serve *static* pages, or pages that couldn't accept user input and change according to user input. However, since then, numerous technologies have been developed to allow dynamic, interactive pages to be served by allowing scripts or programs on the server to generate the pages to be returned to clients for each specific page request. This has allowed the Web to become much more useful, and it's now possible to perform searches, sign up for services, or buy products through the Web, all of which wasn't possible in the age of static Web pages.

CGI

One of the first technologies to be introduced was the Common Gateway Interface (CGI), which is now built into almost all Web servers. For our purposes, CGI is a method of obtaining and returning information to and from the Web server (the official CGI documentation available at http://hoohoo.ncsa.uiuc.edu/cgi/overview.html offers a more precise definition).

CGI, as its name implies, isn't a language, but rather an interface to the Web server. CGI applications can be programmed in a wide variety of languages. Possibly the most common is Practical Extraction and Report Language (Perl), which is known throughout the UNIX world. Perl is interpreted and therefore not very fast. It is, however, a very powerful language, but with that power comes a difficulty level that is significantly higher than that of Visual Basic. Another common language for CGI programming in the UNIX world is C, which offers several performance advantages over Perl, although Perl is definitely more popular than C for CGI application development. CGI in UNIX is not limited to Perl and C; Python is another popular choice. In fact, it's even possible to program CGI applications in UNIX shell script.

Definite advantages of CGI are that it is platform independent and built into most Web servers. So long as a language with an interpreter with multiple-platform support (such as Perl) is used, CGI scripts will run with little change over multiple platforms, such as UNIX and Windows. In Windows, it's possible to develop CGI executables in a variety of compiled environments, including Visual C++ and Delphi, which can access every part of the operating system.

Possibly the greatest pitfall of CGI from a performance point of view in Windows is that each client request for a CGI "Web page" requires a new process to be created, which consumes vast amounts of system resources should there be multiple requests. From the developer's point of view, CGI applications are hard to maintain and very difficult to debug. They also require the developer to perform a significant amount of plumbing, and CGI does not offer many necessary features exposed by other technologies such as ASP.

ISAPI

Microsoft developed Internet Server Application Programming Interface (ISAPI) for its Internet Information Server (IIS) to eliminate the major performance bottleneck of CGI: the need to create a new process for each client request. ISAPI was intended to replace CGI for IIS by allowing developers to do everything they could by using CGI executables, but do it with significantly less performance overhead. ISAPI applications are compiled as Windows dynamic link libraries (DLLs). ISAPI applications run in the IIS process space, which allows for execution faster than that of CGI applications. There are two different types of ISAPI applications, ISAPI Filters and ISAPI Extensions, each with its own specific use.

ISAPI Filters run in the background and can monitor Web server requests and perform a variety of tasks accordingly. Possibly the most well-known example of an ISAPI Filter is the asp.dll filter, which monitors requests for files with the extension ".asp," processes the file using the variables it has access to on the Web server, and returns the resulting page to IIS to send to the client.

ISAPI Extensions are the rough equivalent of CGI executables and run upon client request. ISAPI Extensions have access to important variables pertaining to the request and are typically used to perform database interaction, business logic, and the like.

IDC

Microsoft's Internet Database Connector (IDC) technology has been included in IIS since its inception and is used to provide easy database connectivity for Web pages. Each page requires two files: a query file and a template file. The query file contains the database connection information and the SQL query. The template file contains

HTML and tags denoting where specific data from the query file must be inserted. This method of database connectivity was very easy to use, but it had one major pitfall in that there was no place to put business logic.

ASP

Microsoft combined the best features of IDC and ISAPI to produce Active Server Pages (ASP). ASP pages offer the ease of use of IDC by allowing database and business logic code to be inserted directly into pages using special tags, and they offer the performance advantages of ISAPI. ASP is very scalable, and this has been proven time and time again with many large sites relying on ASP, including Barnesandnoble.com (http://www.bn.com), Fatbrain.com (http://www.fatbrain.com), and Microsoft's huge Web properties (MSN at http://www.msn.com, Microsoft at http://www.microsoft.com, and so on), among many others.

ASP was a huge leap forward and offered a completely new paradigm for dynamic Web page development when it was introduced. ASP's unique approach allowed for dramatic increases in productivity and opened up the exciting area of dynamic Web page development to many novice programmers who were bewildered by the complexity of CGI and ISAPI. This was helped largely by the fact that the primary languages for ASP development are VBScript and JScript, which are very easy to learn and use, especially because VBScript is a subset of the Visual Basic language, and JScript, the Microsoft implementation of JavaScript, is a language commonly used in client-side Web page programming.

ASP's approach spawned a few very similar technologies, most notably JavaServer Pages (JSP). ASP, however, isn't perfect. ASP relies on an interpreted programming model, which affects its performance. A significant amount of plumbing is still required for the programmer to perform in many scenarios—for example, controlling user access to an application. Another problem was exposed while ASP programmers attempted to collaborate with graphic and page design teams. Typically, a site's design is not set in stone and graphic design teams will want to revise designs and make modifications during the design and development phase of a site in a corporate environment. If ASP code is used heavily in pages, this can create a number of complications. Most Web design tools do not recognize ASP code and will either rip it out or distort it. If the design teams hand-code their HTML, they may unwittingly remove, replace, or dislocate code on the page. The obvious solution to this problem is to separate the code from the page, but ASP doesn't allow for this. ASP has numerous other shortcomings, and the list in the following section of the major differences between ASP and ASP.NET details some of the problems and Microsoft's solutions to them.

Differences Between ASP and ASP.NET

ASP.NET isn't simply the next version of ASP—it's a completely redesigned technology that takes the best aspects of ASP and merges them with the power of pure object-oriented programming (OOP), a powerful development framework, to give it a vast range of functionality and the advantages of a fully compiled execution environment. Because the changes between ASP and ASP.NET are so drastic, current

ASP developers must “unlearn” many concepts that they became accustomed to in ASP in order to truly get the most out of ASP.NET.

Compiled Environment

One of the most dramatic changes in ASP.NET is that it’s now a fully compiled environment. Microsoft has implemented this very intelligently and it would be very easy to dismiss ASP.NET as interpreted because, to the programmer and the end user, it appears as if ASP.NET works in exactly the same way as ASP: You modify your ASP.NET page, you refresh the page in the browser, and the changes are reflected. Nowhere are you required to run a compiler. Compilation actually occurs the first time a page is requested after it has been modified. This compiled copy is then kept until the page is modified and requested again. As I’ve already mentioned, this process is totally transparent to the user.

There’s naturally a significant performance advantage of ASP.NET over ASP now, in addition to the obligatory scalability advantage of a compiled application. It’s also important to note that while ASP.NET supports the previously mentioned “compile on demand” functionality, it’s also possible to precompile ASP.NET applications into a .NET DLL, which is the method that Visual Studio .NET uses by default when building Web applications with it.

Based on the .NET Framework

As its name implies, ASP.NET is an integral part of .NET and the .NET Framework. All the core pieces of ASP.NET are implemented as .NET Framework classes. More important, ASP.NET applications have access to the common .NET programming model and the .NET Framework base classes, which include ADO.NET database access, XML parsing, and file IO. This helps make programming ASP.NET applications remarkably similar to programming Win Forms applications or Windows executable applications in .NET. Another advantage of the .NET platform is that it’s completely language independent. I discuss .NET and the .NET Framework shortly.

Server Controls

Microsoft has introduced the concept of server controls in ASP.NET to help simplify Web application development, specifically for Visual Basic programmers who are used to programmatically accessing controls such as labels and text boxes. However, this change also greatly affects traditional ASP programmers, who will have to become accustomed to this new paradigm in Web application programming. Simply put, *server controls* allow programmatic access to HTML elements. This is heavily tied to the next change in ASP.NET.

One of the major elements of plumbing that an ASP programmer must perform is maintaining state. Windows application programmers take state maintenance for granted, as it happens automatically. Basically, *state maintenance* is keeping the values of fields, such as text boxes, when the page is refreshed. Server controls maintain their own state, so this is another plumbing task that the programmer no longer has to perform.

Server controls are designed to be intelligent and take advantage of specific browser capabilities. Web browsers all have different features and capabilities, and server controls will generate code according to what the browser requesting the page can display.

It’s also possible to create your own server controls, which is a great way to encapsulate your code and increase productivity through code reuse.

Event-Driven Programming

Tightly coupled with the server controls paradigm is event-driven programming for ASP.NET pages. The introduction of the event-driven programming model in the Windows programming world revolutionized the way applications were programmed. The introduction of this model into the Web programming world should have an equally great effect on productivity and the simplicity of application development.

Server controls expose events, such as Click, for which event handlers can be programmed. This shields the programmer from having to handle HTTP responses and requests, although it’s still possible to handle them manually using the respective traditional ASP intrinsic objects.

Separation of Code from Content

As I mentioned previously, a major drawback of ASP in certain instances is that there is no way to separate code from page content. In ASP.NET, due to the introduction of server controls and the event-driven model, the trend leans toward placing all the code in event handlers in one place in the file (normally at the top). If this isn’t sufficient, it’s also possible to totally separate code from content by placing all the code in a separate file (VS .NET does this by default).

Session State Management

ASP provided a special collection object where the programmer could store variables pertaining to a particular visitor’s session. This was a commonly used feature, but it had three major shortcomings. First, it wasn’t Web farm–friendly. Session state variables were stored in the IIS process space, and as such, they were tied to a specific machine. Thus, if a visitor to a site had a particular piece of information stored in a session variable on a particular request, and he or she was then sent to another server on the next request, the ASP code would not be able to retrieve the previously stored value. Second, a disadvantage of storing session state in the IIS process space is that should the IIS service have to be restarted, the session information is lost. The third problem was slightly more subtle, but important nonetheless: ASP kept track of visitor IDs by using cookies, which aren’t supported by all browsers, or are not guaranteed to be enabled in all user browsers.

ASP.NET offers solutions for all the problems presented. First, it’s now possible to store session state in a SQL Server database, which is naturally accessible to all the servers in a Web farm. Second, it’s possible to store session state out of process, which alleviates the problem of IIS service restarts. Third, ASP.NET now allows for cookie-less session state management.

User Input Validation

A major aspect of plumbing that ASP requires is that of *input validation*—checking that information that users have entered is in the correct form. ASP.NET provides a set of server controls that make this task almost trivial. These controls are intelligent and produce client-side validation code for supporting browsers and server-side code for the rest.

Web Services

Web Services are a large part of ASP.NET and .NET as a whole. *Web Services* are classes, or objects, that expose certain methods and make them accessible over the Internet. Web Services are based entirely upon open standards such as Simple Object Access Protocol (SOAP), XML, and HTTP, thus allowing interoperability with “Web services” on other platforms, such as UNIX, as well as those built using the Visual Basic 6 SOAP kit. Naturally, the Web Services model in .NET allows for the consumption of Web Services as well. Web Services should prove to be an exceptionally useful technology in the areas of business-to-business (B2B) communication and commerce, and business-to-person (B2P) commerce.

Caching Services

ASP.NET offers two primary caching techniques: data caching and output caching. *Data caching* allows programmatic access to a cache store where objects and values can be stored. *Output caching* allows caching of the output of specific pages for specific periods of time. Previously, the functionality of data caching had to be developed from scratch by the developer, and output caching would also have to be separately developed and implemented. Naturally, these caching features allow for greater performance advantages.

Security

Security usually plays a large role in a Web application. In ASP, it was up to the developer to implement his or her own security mechanisms, especially in the area of user authorization. ASP.NET features built-in security configuration in its application settings files. This takes a lot of tedious, yet necessary, work off the hands of the developer.

Debugging and Tracing

Every programming environment is constantly striving for better, more advanced, and easier debugging tools and methods. Web application developers have always had to deal with very rudimentary tools, with functionality and usability far behind that of debugging tools for Windows applications developers. ASP.NET and Visual Studio .NET aim to solve some of these developer woes by providing more advanced debugging tools. *Tracing* allows developers to inspect and gather important application performance information. Tracing also allows developers to check whether control flow is executing correctly. Developers can now make use of the

Output window in VS .NET, a commonly used tool familiar to Visual Basic developers. ASP.NET also provides more detailed and useful error messages, in addition to a .NET stack dump that details the exact steps of the compiler in relation to the classes being used and the methods being executed.

Deployment

Deployment has traditionally been a huge headache in the Web application development process. Complicated applications required numerous operations to be performed before they would run. Compounding this problem was the requirement of IIS service restarts for updates to business logic components and the mandatory registration of these COM components. ASP.NET no longer uses COM components. Component registration is no longer required. DLLs are no longer locked, and if a component DLL is replaced, all the sessions currently using the old one will continue to use it and all future sessions will use the new one. IIS service restarts are also a thing of the past.

What Is VB .NET?

Microsoft has recognized that Web application development is currently playing and will continue to play an important role in the lives of developers. They also recognize that many of the 7-million-plus current Visual Basic developers aren't prepared to dabble with Web development due to the time investment that it takes to get up to speed (prior to ASP.NET). One of the primary goals of ASP.NET is to make Web application development as similar to Windows application development as possible without sacrificing power or performance, while still making the skills that ASP developers had learned pertinent. Naturally, Visual Basic .NET is a very important part of this challenge, as the primary audience to help start Web development was Visual Basic developers. Another factor was that the primary language used in ASP development was VBScript, a subset of the Visual Basic language, and these ASP/VBScript programmers could not be alienated.

VB .NET is the successor of the Visual Basic language. It is, as its name implies, a .NET Common Language Specification (CLS)–compliant language. There have been some fairly major changes from Visual Basic to VB .NET, most notably the inclusion of full OOP support (a prerequisite of the CLS). When you refer to Visual Basic, people normally assume that you're talking about the development environment as well as the actual language. VB .NET is a stand-alone .NET language, and it's shipped with the .NET Framework SDK. This only includes the compiler, not the IDE. However, VB .NET can be run as part of a new global development environment, Visual Studio .NET, which is a commercial product.

VB .NET As a Language

As previously mentioned, VB .NET is a .NET CLS-compliant language. It's distributed as part of the .NET Framework, and no additional downloads are necessary. There are several significant differences between VB .NET and previous versions of Visual Basic. VB .NET is fully typed and has full support for inheritance, polymorphism, and encapsulation, among other important OOP concepts. Here’s a brief list of some of the other changes implemented in VB .NET:

- No more variants
- Class oriented
- Try..Catch..Finally error handling
- Support for interfaces
- Support for delegates
- Reliance on the .NET Framework

The bane of all reasonably knowledgeable programmers, the Variant data type, has finally been removed from the Visual Basic language. This is in accordance with the full type-safety goal of VB .NET. Although not specifically required by the compiler, VB .NET also encourages all type conversions to be performed explicitly, which makes type-related errors much easier to detect, especially for less experienced programmers.

Classes are a very important part of VB .NET. Inheritance works by starting off with a class and modifying it. Thus, for inheritance to be properly supported in VB .NET, classes must play a pivotal role, which they do. All applications are implemented as classes. All Web Forms are classes. All Win Forms are classes. All Web Services are classes. All user controls are classes. Most functionality—file IO, data access, and so on—in the .NET languages is implemented through the base classes. It’s very important, therefore, to understand what a class is, and this will be dealt with briefly in the next chapter.

VB .NET finally gives Visual Basic programmers what many have been requesting for a long time: powerful error handling structures (the .NET Framework also provides some powerful exception handling classes). The Try..Catch..Finally structure allows precise handling of specific errors in particular pieces of code. Gone are the days of the awful “On Error Goto Label” structure (although it still does exist for the purposes of backward compatibility).

Again, in line with OOP compliance, VB .NET supports interfaces and allows for interfaces to be created and implemented. Support for delegates has also been added.

Possibly the most dramatic change to the Visual Basic language, and the one that is most likely to affect programmers the most, is the reliance of VB .NET on the .NET Framework. Most aspects of the language have changed because of this—from OOP support, to data access, to debugging, to compilation, to deployment. In Win Forms applications, all controls are .NET Framework classes. In Web Forms applications, all server controls are .NET Framework classes. In fact, both types of applications are inherited .NET Framework classes themselves. ADO’s successor, ADO.NET, which provides data access, is implemented as .NET Framework classes. XML parsing is implemented as .NET Framework classes. Even Win32 API functions are implemented as .NET Framework interop classes. Thus, it’s extremely important that all developers planning to use ASP.NET, or any other .NET application model, have a thorough understanding of the .NET Framework.

VB .NET IDE

In .NET, all languages share a common Visual Studio IDE. Microsoft chose to do this to aid cross-language development, which is very possible in .NET. For previous Visual Basic developers, it's easy to set up VS .NET to look very similar to the default Visual Basic layout. Likewise, ASP developers who used Visual InterDev can change the VS .NET environment with the click of a button.

VS .NET adds a few very useful features to what VB and VI developers know. Possibly the most helpful is the Server Explorer, a dockable tree view containing information about machine resources, including databases, database connections, event logs, and performance counters. This description makes it sound fairly superficial but, although naturally everything that can be done in it could previously be done by using external tools (such as the SQL Server Enterprise Manager and the Windows Administrative tools), a huge productivity boost is gained by providing scaled-down versions of these tools in the IDE, which provides most of the functionality that you'll require so—you'll rarely have to load up the external tools.

The Toolbox has been completely rebuilt and contains different tools according to the type of application being built. As previously mentioned, it's important to understand that all the controls available, no matter what application you're building, are actually .NET classes.

VS .NET is a complete IDE for both Win Forms and Web Forms application development. Again, as previously mentioned, Microsoft has spent a great deal of time making Web application development similar to traditional Windows application development, and although the IDE's form designer does change depending on the application being developed, both Win Forms and Web Forms user interfaces are made by dragging and dropping controls from the toolbox.

The help system in VS .NET is completely integrated. It is a dynamic, HTML help system, which changes the displayed topics according to the current task being performed (or code being typed).

A task list has been added, which not only provides the obvious functionality of allowing tasks "to do," but also automatically adds entries based on errors in the code.

The new, tabbed window manager makes swapping between code, form design, help, and other main views much faster and easier by providing a tabbed bar above the windows with all the entries on it.

The code editor has been given a significant overhaul to help boost productivity. Microsoft pioneered the red "squiggle" in Word to highlight spelling mistakes and later introduced the green variety for grammatical errors. VS .NET now features a similar concept for programming: the blue "squiggle." Anything from invalid references to omitting type conversions is highlighted and added to the task list. This replaces the pop-up message box that Visual Basic programmers know so well. Hovering the mouse over the code in question will result in a tool tip detailing the problem. The editor has a few more useful productivity additions. Automatic completion of code blocks (If…End If and forth) and automatic code indentation are also noteworthy additions.

A fairly large change for Visual Basic programmers is the removal of the option to view only one sub at a time. Microsoft has, however, added a new feature to replace this: collapsible code. All functions and subroutines can be "collapsed" so that only the declaration is visible. It’s also possible to create code “regions” for collapsing entire sections of code (normally multiple functions and subs). The collapsing/expanding is done via tree view–style handles.

The AutoComplete technology in VS .NET has been upgraded to extend to XML files and DHTML, in addition to regular Visual Basic code. This offers another great productivity boost for developers who prefer to code their Web pages by hand or use XML.

The .NET Framework

To understand what the .NET Framework is, and what it does, it’s important to know why Microsoft created it and what Microsoft had in mind when it created it. The .NET Framework is core of Microsoft’s greater .NET “software as a service” vision. Simply put, the .NET Framework is the back end of .NET. It’s what developers use to build .NET applications and what these applications run on top of. The .NET Framework consists primarily of a set of base classes, which expose the functionality required to build and run any type of .NET application, be it a Web application, a Windows application, or a Web Service. The .NET Framework also contains the core .NET infrastructure for running these applications, such as compilers and runtime services, such as automatic garbage collection. Additionally, full-scale deployment facilities are available.

Microsoft created the .NET Framework for numerous reasons. Some of the most important reasons are as follows:

* To create a complete development framework that’s entirely integrated, from development to code execution
* To eliminate language dependence for employing different technologies (e.g., C++/Java-style languages could previously not be used to create ASP Web applications)
* To allow complete language interoperability (cross-language inheritance and so forth)

The .NET Framework provides a rich environment for development, debugging, deployment, and execution. Because all parts of the development cycle now belong to the same system, the developer can gain a significant productivity boost. This can be attributed mainly to the fact that code in one language no longer has to be rewritten to be used in another language; that all languages use the same programming classes, so tasks such as file I/O, XML parsing, and data access are identical in all languages (barring language syntax differences); and debugging, execution, and deployment all operate identically and expose the same features, so the developer doesn’t need to learn a new environment for each language that he or she uses.

> **NOTE** Although it’s not a part of the .NET Framework, which is available freely, Visual Studio .NET has been developed with the same goals in mind. As a result, developers can create .NET applications even more productively by using Microsoft’s .NET programming IDE, VS .NET.

The Common Language Runtime

The common language runtime (CLR) is the execution engine for all .NET applications. It's the part of the .NET Framework that performs all the compilations and provides runtime services to the running .NET application. Here's a brief list of features pertinent to ASP.NET developers:

- Code management (loading and execution)
- Conversion of IL to native code
- Verification of type safety
- Exception handling
- Interoperation between managed and unmanaged code (COM objects and so forth)
- Memory management for managed objects
- Support for developer services such as debugging

Most of these services are self-explanatory; however, a couple of them do require clarification. The conversion from IL to native code is a fairly involved topic that I'll deal with shortly. The .NET Framework insists that all language are completely type safe. The CLR ensures this. COM objects and non-.NET DLLs can be used in .NET applications for the sake of code reuse and backward compatibility, but because they were created in non-.NET languages, the CLR assumes that their code is unsafe and thus treats them strictly (e.g., disallowing direct memory access).

As I've already discussed, the .NET Framework is language neutral. Microsoft has created a specification, the Common Language Specification (CLS), that outlines the requirements that .NET CLS-compliant languages must meet so that vendors can create other languages for or port other languages to .NET. This language neutrality is possible mainly due to the common execution engine (the CLR) and that most tasks are performed using the common .NET base classes. There's also a negligible performance difference between languages, so long as the .NET languages are developed to be efficient (naturally, all the .NET languages that Microsoft has developed are very efficient and offer almost identical performance in most areas where previously one particular language may have stood out).

The languages currently available number over 30; however, the following are the most well-known and prominent releases.

Microsoft languages:

- Visual Basic .NET
- C#
- Managed Extensions for C++
- JScript .NET
- J#

Third-party languages:

- Perl
- Python

* Eiffel
* COBOL

Microsoft Intermediate Language (MSIL or IL) is a CPU-independent language that .NET applications are compiled to before deployment. Its instruction set has been developed specifically for the .NET Framework and the .NET base classes. It’s a very plain language that can roughly be equated to a cross-platform ASM for using the .NET Framework. The following is a sample of MSIL code taken from the HTMLButton class’ PreRender event:

```
.method family hidebysig virtual instance void
       OnPreRender() il managed
{
  // Code size       44 (0x2c)
  .maxstack  8
  IL_0000:  ldarg.0
  IL_0001:  call       instance class [System]System.ComponentModel.EventHandlerLi
st System.Web.UI.Control::get_Events()
  IL_0006:  ldsfld     class System.Object System.Web.UI.HtmlControls.HtmlButton::
EventServerClick
  IL_000b:  call       instance class [mscorlib]System.Delegate [System]System.Com
ponentModel.EventHandlerList::get_Item(class System.Object)
  IL_0010:  ldnull
  IL_0011:  call       bool [mscorlib]System.Delegate::op_Inequality(class [mscorl
ib]System.Delegate,
                                                         class [mscorl
ib]System.Delegate)
  IL_0016:  brfalse.s  IL_002b
  IL_0018:  ldarg.0
  IL_0019:  callvirt   instance class System.Web.UI.Page System.Web.UI.Control::ge
t_Page()
  IL_001e:  brfalse.s  IL_002b
  IL_0020:  ldarg.0
  IL_0021:  callvirt   instance class System.Web.UI.Page System.Web.UI.Control::ge
t_Page()
  IL_0026:  callvirt   instance void System.Web.UI.Page::RegisterPostBackScript()
  IL_002b:  ret
} // end of method HtmlButton::OnPreRender
```

As you can see, programming in MSIL directly would be very time consuming and isn’t desirable. This is what the .NET language compliers compile their high-level code to. The CLR isn’t involved in this step. Depending on the type of application, the CLR will compile this IL code to native code (for example, x86 Win32 code) either on installation (Win Forms) or on the first time a page is accessed since modification in ASP.NET. MSIL code is not interpreted.

Base Classes and Class Hierarchy

At first, the whole .NET Framework concept can be quite confusing. The area that's possibly the most confusing is the base classes. It is, however, an extremely well-structured and logical system that will greatly increase your productivity, so it's important for you to understand it.

Several terms that I'll use regularly in the book and that are important to understand are as follows:

Assembly: Assemblies are a core part of all .NET applications. They provide the CLR with information about the content of the application. Some of the information that assemblies contain that is pertinent to ASP.NET development includes the following:

* The identity and version of the assembly
* The files that the assembly consists of
* The resources and types that the assembly consists of
* A list containing dependencies
* The compiled IL code

Namespace: Namespaces in .NET allow for logical naming patterns of classes. For example, you may assign the namespace name "MyClass" to a class and then assign the namespace name "MyClass.SomeFunctionality" to a subclass of it. Namespaces are not in any way connected to assemblies. Assemblies can contain many namespaces, and namespaces can extend over many assemblies.

Class: Classes are a very simple concept, and you can find them in most modern languages. For those who have not previously come across them, classes are simply data structures with associated functionality.

Several namespaces are particularly pertinent to ASP.NET development. Naturally, the list of namespaces that follows is far from exhaustive. You can find a longer list in Appendix A.

* *System.Web:* This namespace contains classes that allow for communication between the server and the Web browser. It includes classes for many tasks, including cookie manipulation and file transfer.
* *System.Web.UI:* This namespace contains classes for manipulating the output to the browser.
* *System.Web.Util:* This namespace contains classes for miscellaneous functions, such as sending e-mail and adding EventLog entries.
* *System.Web.Services:* This namespace contains classes for building and using Web Services.
* *System.Xml:* This namespace contains classes for XML processing and manipulation.
* *System.IO:* This namespace contains classes for file I/O.

* *System.Data:* This namespace contains classes for database access using ADO.NET, using both the SQL Managed Provider and the ADO.NET Provider.

In summary, the .NET Framework is a rich programming environment that has been designed to solve the problems of the past and improve developer productivity in all facets of application development, from development to debugging to deployment.

Other Parts of Microsoft .NET Relevant to Developers

Microsoft's .NET strategy is very large and has an effect on almost every single product that Microsoft makes. As I mentioned previously, Microsoft's .NET strategy can be summed up quite simply: "software as a service." Again, the .NET Framework is the infrastructure for building .NET applications. However, there are other parts of Microsoft .NET pertinent to the developer.

VS .NET

Although the .NET Framework provides a very powerful and productive infrastructure for developing applications, it provides no user interface or IDE for harnessing these strengths effectively. VS .NET is an IDE developed specifically and only for .NET application development using the .NET Framework. It's an extremely powerful and productive development environment, with features such as full cross-language support, integrated debugging, AutoComplete technology, and full support for Web Forms, Win Forms, and Web Services application development. It's a highly extensible environment and it allows for .NET languages other than those developed by Microsoft to be used in it, with the same features.

Enterprise Servers

Although not explicitly tied to .NET and .NET applications, Enterprise Servers do complement .NET applications, and as such it's useful to be aware that they exist. Here's a list of the relevant packages:

* BizTalk Server 2000
* SQL Server 2000
* Internet Security and Acceleration Server 2000
* Exchange Server 2000

BizTalk Server, simply put, is a product designed to help companies build and deploy integrated business processes, internally and with their trading partners. It provides features such document routing and transformation. BizTalk Server relies on open standards, such as XML, XSLT, HTTP, HTTPS, SMTP, and SOAP.

SQL Server is a high-performance, high-scalability database server. It has intrinsic support for XML and includes powerful administration and monitoring tools. SQL Server will most likely play a pivotal role in your application development if databases are involved.

Internet Security and Acceleration Server provides both firewall security and caching services to Web applications. It has an advanced management interface for administration and monitoring for both the firewall and caching services that it provides.

Exchange Server is an advanced collaboration system intended primarily for use as a large-scale corporate e-mail system. It provides full integration with both Windows 2000's Active Directory system and Outlook Web Access. Microsoft has also innovated the Web Storage System, which allows for messages and documents to be stored and accessed through the Web. Integration with Web Forms is also possible.

Blackcomb

"Blackcomb" is the code name for the 2005 release of the Windows desktop operating system in which Microsoft aims to further their .NET vision of "software as a service" such that most software for Blackcomb is provided through the Internet as a service using .NET. Although there will likely be new releases of the .NET Framework and VS .NET between now and the release of Blackcomb, the current version of .NET will remain largely the same in structure and functionality and forms the base for programmers using Microsoft technologies for the foreseeable future.

Summary

In the primitive days of Web application programming, developers were required to spend large amounts of time learning complex systems, as they were required to do most of the "plumbing" of the development. Microsoft pioneered the way to simpler, more productive, richer application development with ASP. ASP.NET takes the brilliant concepts of ASP and integrates them as a core part of a completely rebuilt application programming model designed specifically with developer productivity, ease of use, and ease of transition from Windows application programming in mind.

ASP.NET is a core part of the new .NET Framework, an entirely new, centralized, unified, programming language–independent application development model for Windows applications, Web applications, and Web Services.

The .NET Framework is the development infrastructure behind Microsoft's "software as a service" .NET vision, which encompasses and affects almost all Microsoft products. New versions of software products will be released to integrate with .NET Framework applications more tightly, such as Windows .NET.

Visual Studio .NET is a powerful IDE for .NET application development, which has full, integrated support for all the .NET Framework features that are exposed, such as cross-language development and integrated debugging.

Now that you have a decent understanding of what .NET programming is all about, it's time to install the software. The next chapter guides you through the setup of Visual Studio .NET and all the other related software components that are prerequisites for ASP.NET development.

Chapter 2

Installation

Visual Studio .NET isn't simply a stand-alone product for building Web applications—in fact, it relies on several entirely separate software components to provide an end-to-end solution for designing, implementing, and testing software programs. As I touched on in Chapter 1, there's a distinction to be made between VS .NET and the .NET Framework SDK. Whereas VS .NET is simply a development environment, the .NET Framework contains the actual runtime engine and related components for compiling and executing .NET applications, including ASP.NET applications. However, in the case of ASP.NET, the .NET Framework is not simply a stand-alone platform for running ASP.NET either: It relies heavily on Microsoft Internet Information Server (IIS) for providing the basic functionality of serving data to clients on the Internet over HTTP.

Internet Information Server

IIS has been an integrated component in the Windows NT line of operating systems for several years now, and Windows 2000 (all editions), Windows XP Professional, and Windows .NET Server (all editions) ship with IIS version 5.0 or above, which is a requirement for developing and running ASP.NET applications. IIS' core purpose is that of an HTTP server, which in essence means that IIS is responsible for receiving requests for Web pages from clients on the Internet, processing these requests, and then returning the pages to the clients.

> **NOTE** There's obviously a lot more to IIS than simply sending pages to clients over the Internet. It is, in fact, a fairly complex piece of software; the details of how IIS works and all the services that it provides warrants an entirely separate book.

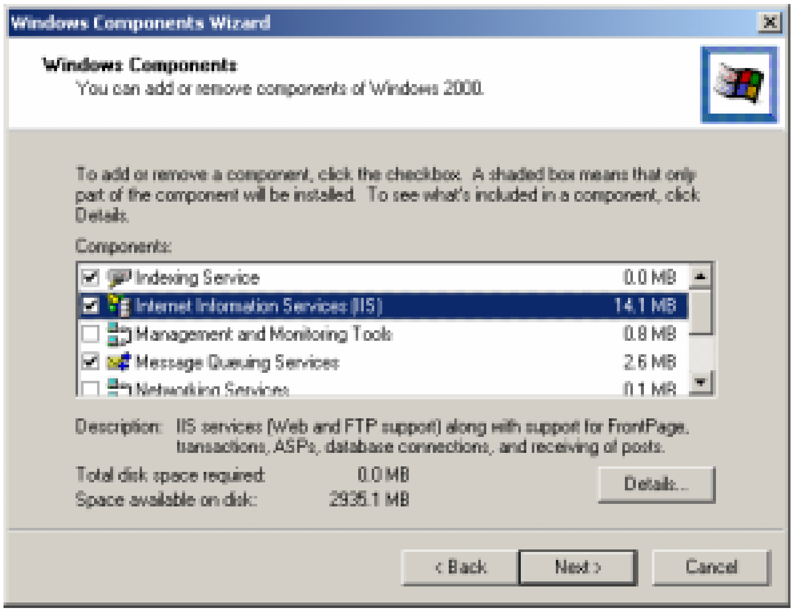
ASP.NET integrates into IIS, allowing IIS to handle the receiving of requests and the final sending of data to clients. But during the processing of a request, IIS will pass all the information known about the request to ASP.NET, which will then perform its own processing of the requested page and finally pass the resulting data back to IIS to send to the client. IIS therefore plays a crucial role in developing ASP.NET applications, as it's the component that provides the underlying groundwork for communications between the user and the server.

Installing IIS

Because ASP.NET needs to be set up within IIS, it's important that IIS is installed before ASP.NET. If you're running a server version or edition of Windows (such as Windows 2000 Server or Windows .NET Server), IIS was probably installed by default. However, for other editions, IIS is most likely not installed and before you proceed, you'll need to manually install it.

Installing IIS is a very simple process. Because it's a Windows component, it's added (or removed) through the Control Panel's Add/Remove Programs applet. The exact steps may differ slightly by operating system, but the following general steps apply to all of them:

1. Open the Control Panel's Add/Remove Programs applet.
2. Click the Add/Remove Windows Components button. Figure 2-1 shows the dialog box that should appear.



Windows Components
You can add or remove components of Windows 2000.

To add or remove a component, click the checkbox. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.

Components:

☑	Indexing Service	0.0 MB
☑	Internet Information Services (IIS)	14.1 MB
□	Management and Monitoring Tools	0.8 MB
☑	Message Queuing Services	2.6 MB
□	Networking Services	0.1 MB

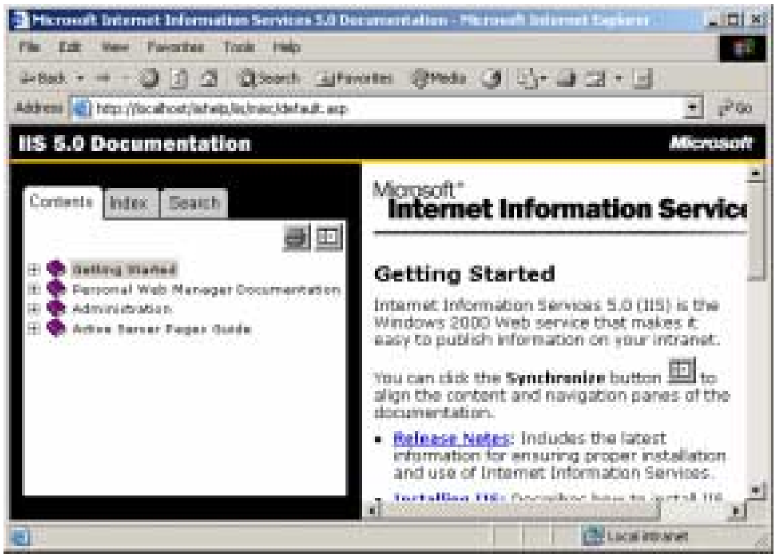
Description: IIS services (Web and FTP support) along with support for FrontPage, transactions, ASPs, database connections, and receiving of posts.

Total disk space required: 0.0 MB
Space available on disk: 2935.1 MB

Insert 0015f0201.tif

Figure 2-1. The Add/Remove Windows Components Wizard

3. If the check box next to the Internet Information Services (IIS) option is checked, IIS is already installed so you can click Cancel. However, if the Internet Information Services (IIS) check box isn't checked, check it and click Next to proceed.
4. You may be asked to insert your Windows installation media at this point, and Windows will then install IIS. Once this is finished, dismiss the wizard by clicking Finish and close the Add/Remove Programs applet.
5. To ensure that IIS has been installed correctly, open Internet Explorer and browse to the following URL: http://localhost/iishelp. Figure 2-2 shows what should display if you're running Windows 2000 (IIS 5.0), although this URL should display a Help screen when you're using IIS 5.1 (Windows XP) or IIS 6.0 (Windows .NET Server) as well.


Microsoft Internet Information Services 5.0 Documentation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/iishelp/iis/misc/default.asp
Go
IIS 5.0 Documentation
Microsoft
Contents Index Search
Getting Started
Personal Web Manager Documentation
Administration
Active Server Pages Guide
Microsoft® Internet Information Servic
Local intranet

Getting Started

Internet Information Services 5.0 (IIS) is the Windows 2000 Web service that makes it easy to publish information on your intranet.

You can click the **Synchronize** button to align the content and navigation panes of the documentation.

- Release Notes: Includes the latest information for ensuring proper installation and use of Internet Information Services.
- Installing IIS: Describes how to install IIS

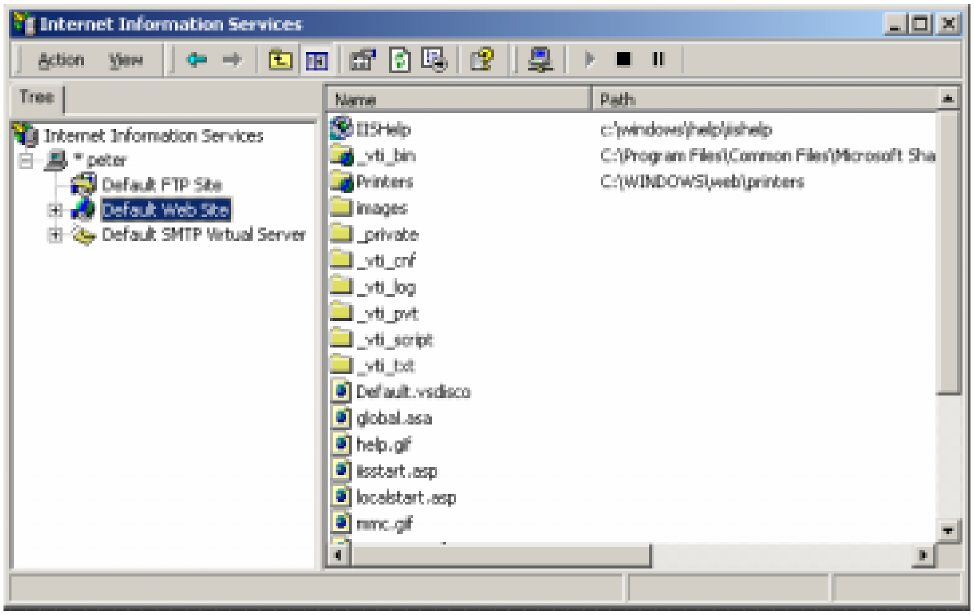
Insert 0015f0202.tif

Figure 2-2. The IIS Help Web site, which is installed by default during the IIS setup process

If the screen in Figure 2-2 does appear, IIS has been installed successfully. However, if an error such as “Cannot find server” appears, it’s likely that the installation failed, in which case you should try uninstalling and then reinstalling IIS; failing that, you should contact one of your local Microsoft support services for help.

An IIS Primer

Because IIS is a Windows Service that runs in the background, it isn’t a front end, as such. However, when IIS is installed, another applet is added to the Administrative Tools section of the Control Panel: the Internet Services Manager (ISM), which is an MMC plug-in that can be used to configure IIS. This is a very useful and important tool, and although you shouldn’t need to modify the default settings of IIS, understanding the basics of IIS is crucial when you build ASP.NET applications, especially when you plan to deploy them to a “live” (or “production”) Web server. Figure 2-3 shows the ISM after it has been loaded from the Control Panel.



Internet Information Services
Action
View
Tree
Internet Information Services
* peter
Default FTP Site
Default Web Site
Default SMTP Virtual Server

Name	Path
IISHelp	c:\windows\help\iishelp
_vti_bin	C:\Program Files\Common Files\Microsoft Sha
Printers	C:\WINDOWS\web\printers
images	
_private	
_vti_cnf	
_vti_log	
_vti_pvt	
_vti_script	
_vti_txt	
Default.vsdisco	
global.asa	
help.gif	
iisstart.asp	
localstart.asp	
mmc.gif	

Insert 0015f0203.tif

Figure 2-3. The Internet Services Manager

The Home Directory

The simplest way to think of IIS is as a means to share files over the Internet. Just as with sharing files in Windows over a Windows network, in IIS you need to choose a folder whose files are shared, as you obviously don't want to share your entire computer's data. By default, this folder is C:\Inetpub\wwwroot\.

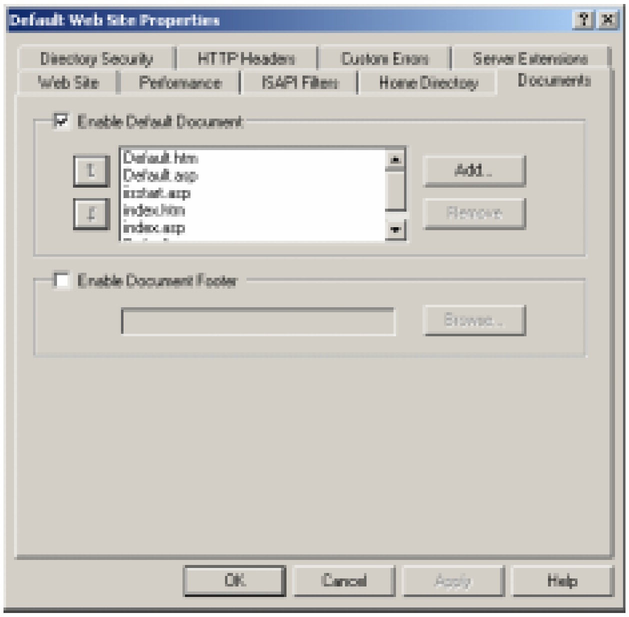
In Figure 2-3, the files listed in the right pane are the files created in the C:\Inetpub\wwwroot\ directory during the installation of IIS. These are simply a few HTML documents (if you're unfamiliar with HTML, it's introduced in Chapter 3) and related images that act as an introduction to IIS. When you visit http://localhost in Internet Explorer (IE), IE effectively sends a message to IIS asking for the list of files in the "home directory" (the reason this doesn't actually happen will be covered shortly). To access a particular file in the home directory in IE, the filename must be included in the URL after the initial http://localhost. For example, if there was a file called readme.txt in the home directory, you could access it in IE by browsing to http://localhost/readme.txt. Likewise, if there was a file called hello.txt inside a subdirectory named newfolder of the home directory (in other words, C:\Inetpub\wwwroot\newfolder\hello.txt), the URL to access it would be http://localhost/newfolder/hello.txt.

You can test out this concept for yourself by creating text files and putting them into C:\Inetpub\wwwroot (or subdirectories thereof) and accessing them through IE. If you're on a network, you can access these files from other computers by substituting http://localhost with http://ipaddress, where ipaddress is the IP address of your machine on the network (such as 192.168.1.10).

The Default Document

When you visit a site such as http://www.microsoft.com, you don't get a list of files—you're sent an HTML page. If you visit http://www.microsoft.com/net, you don't get a list of files either—you're also sent an HTML page. In fact, it's very rare that an HTTP server will return a list of files to a user or client on the Internet, for a couple of reasons. First of all, it'd be very user-unfriendly to expect a user to select the file that he or she would like to view when he or she visits a site. The fact that the Web site is even made up of files should be abstracted from the user, which is why we have hyperlinks ("links") on Web pages. Second, from a security perspective, giving the user a list of files in a directory on a Web server is far from good practice, and can often give an attacker information that helps him or her compromise the server.

To counter these issues, IIS includes support for a *default document.* This is basically the file that IIS will return by default if the user request doesn't specify a specific file to be returned. For example, if the default document is set to be index.htm, and the user submits a request for http://localhost, the browser will in fact return http://localhost/index.htm if it exists. IIS implements this functionality as a list of "default filenames," so that it's possible to specify that if a directory is requested by the user, IIS will attempt to return the first file in the list, and if that does not exist, then the second file, and so on. To configure the list, right-click the Default Web Site node in the ISM, click Properties, and go to the Documents tab, as shown in Figure 2-4.



☑ Enable Default Document

Default.htm
Default.asp
iisstart.asp
index.htm
index.asp

☐ Enable Document Footer

Insert 0015f0204.tif

Figure 2-4. The property pages for the Default Web Site

This is the reason why when you visit http://localhost, an HTML page is returned rather than a list of files. If you look in the home directory, you'll find that there is in fact a file named using one of the prescribed default document names in the Default Web Site Properties dialog box.

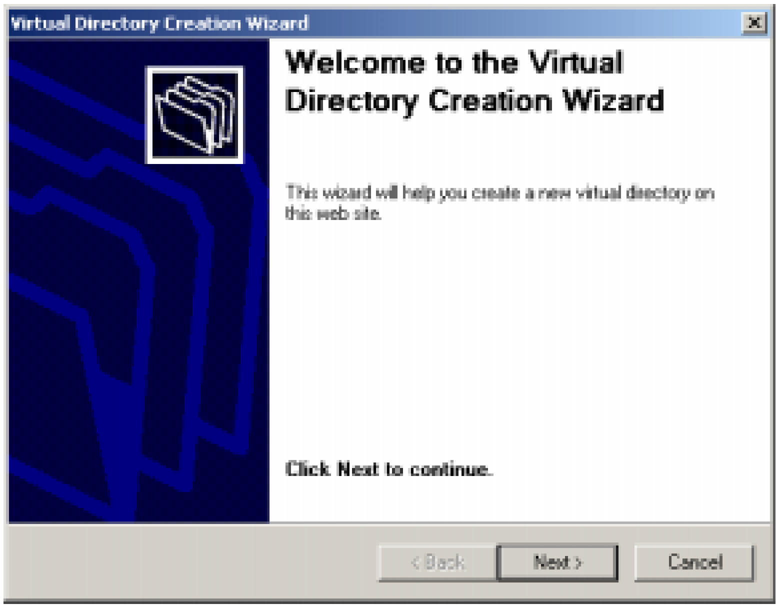
Virtual Directories

An important concept to understand in IIS (because ASP.NET relies on it so heavily) is that of a *virtual directory*. It’s conceivable that you may want to develop and test numerous ASP.NET applications on a single server, and IIS allows for this. If this were not the case, you’d have to have a new server for every application you developed.

Assume that you have numerous Web sites that you’re developing on your machine, but they’re all in different locations. For example, one might be stored in C:\MyTestSite, and another might be in C:\Widgets\Website. The problem presented is that there is no one specific “home directory” underneath which all of the Web sites reside—they’re all stored in disparate locations on the machine. How then can IIS be set up so that all the Web sites can be accessed via http://localhost/? The solution is virtual directories.

To demonstrate the concept, create a new folder, C:\MyTestSite, and place a plain text file called hello.txt in it. Then follow these instructions:

1. Right-click the Default Web Site node in the ISM and click New ➢ Virtual Directory. The Virtual Directory Creation Wizard should appear, as shown in Figure 2-5.



Welcome to the Virtual Directory Creation Wizard

This wizard will help you create a new virtual directory on this web site.

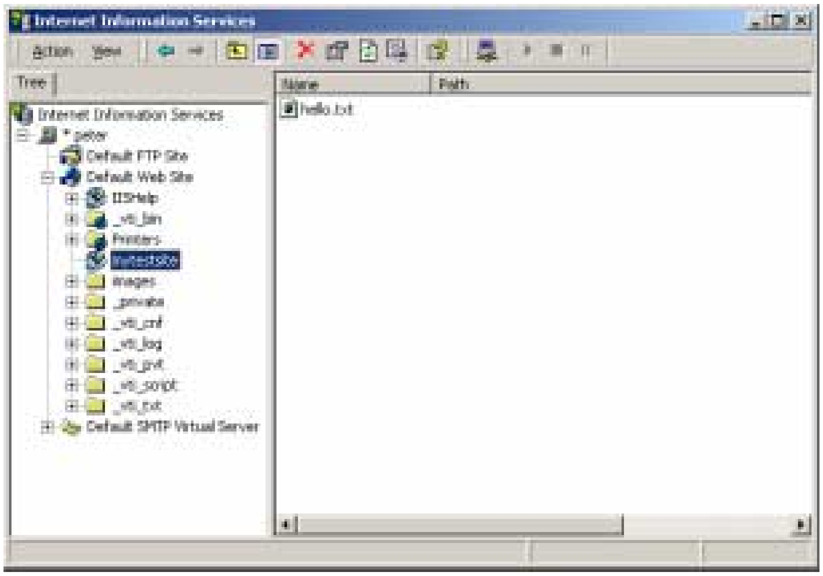
Click Next to continue.

Insert 0015f0205.tif

Figure 2-5. The Virtual Directory Creation Wizard

2. Click Next, and in the following screen, type **mytestsite** in the Alias text entry field.
3. Click Next and enter **C:\MyTestSite** in the Directory text entry field.
4. Click Next, accept the defaults in the following screen, and click Next again.
5. Click Finish.

The virtual directory has now been created, and Figure 2-6 shows IIS with it selected.



Insert 0015f0206.tif

Figure 2-6. The ISM with the newly created virtual directory selected

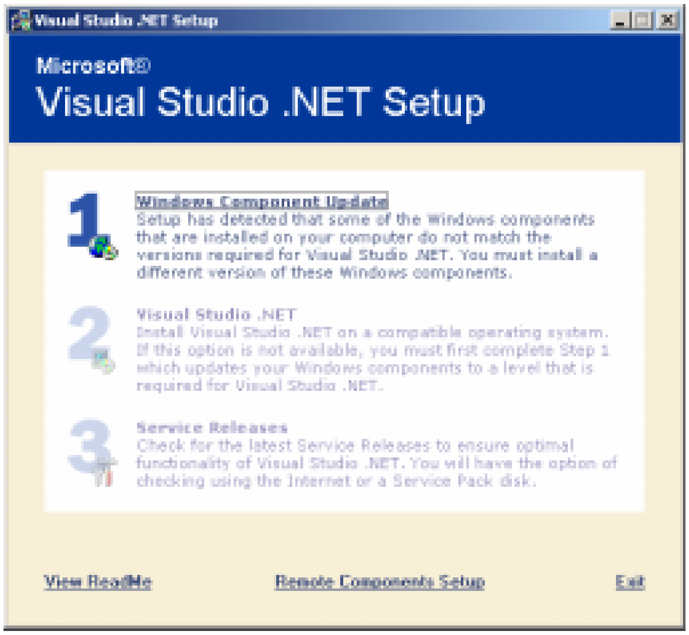
Notice how the virtual directory's icon is different from the actual subfolders in the home directory. The alias that you chose when creating the virtual directory describes how the virtual directory is named and accessed. A virtual directory is simply accessed as if it were a normal subfolder of the home directory. For example, to access the hello.txt file in IE, you would use http://localhost/mytestsite/hello.txt as the URL—the virtual directory looks and behaves exactly as if it were a regular directory, and the user is none the wiser.

However, although this use of virtual directories can be very helpful, they do also serve another different, yet very important function: They inform IIS that the content of the virtual directory is an entirely new Web application, as opposed to a regular directory that's assumed to be part of the Web site or application being hosted in the home directory. Although the importance of this may not be clear now, as you begin to build and test your ASP.NET applications, it should become so, especially in the last chapter of this book, which deals with the deployment of ASP.NET applications to real, live Web servers. For now, all you need to understand is that for each ASP.NET application that you build, a new virtual directory will automatically be created for it that identifies it to IIS as an entirely separate application.

Installing the .NET Framework and Visual Studio .NET

As I've mentioned previously, ASP.NET is not inextricably bound to Visual Studio .NET, and it can operate as a stand-alone product as part of the .NET Framework. Therefore, the installation of VS .NET takes place in two parts: First you install the .NET Framework (and its dependencies), and then you install VS .NET. Fortunately, the VS .NET installer handles the process seamlessly and for the most part requires very little user intervention beyond confirming choices that it has made.

To begin the installation, insert the first VS .NET installation CD or the VS .NET installation DVD. If you're installing from CD media, you'll be prompted to switch CDs throughout the process. The Visual Studio .NET Setup program should start automatically, as shown in Figure 2-7.



Visual Studio .NET Setup

Microsoft®
Visual Studio .NET Setup

1 Windows Component Update
Setup has detected that some of the Windows components that are installed on your computer do not match the versions required for Visual Studio .NET. You must install a different version of these Windows components.

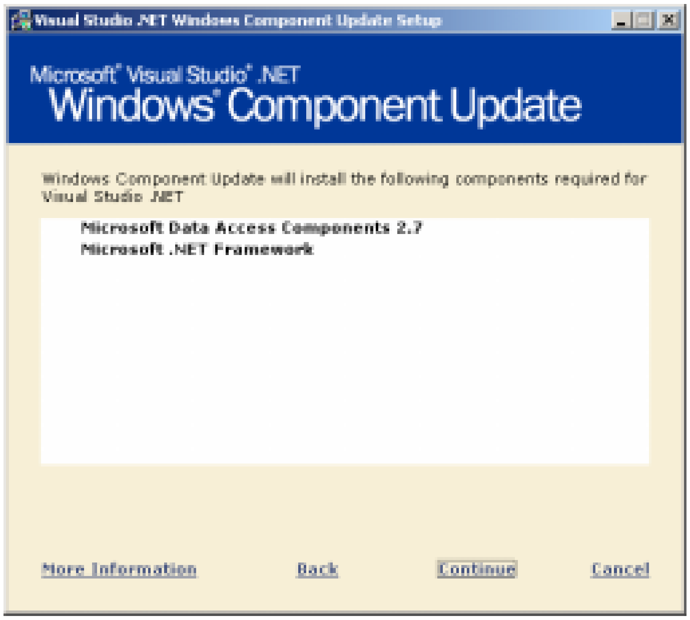
2 Visual Studio .NET
Install Visual Studio .NET on a compatible operating system. If this option is not available, you must first complete Step 1 which updates your Windows components to a level that is required for Visual Studio .NET.

3 Service Releases
Check for the latest Service Releases to ensure optimal functionality of Visual Studio .NET. You will have the option of checking using the Internet or a Service Pack disk.

Insert 0015f0207.tif

Figure 2-7. The Visual Studio .NET Setup menu

The first step is to install both the .NET Framework dependencies and the .NET Framework itself. Click option 1, Windows Component Update. The Windows Component Update setup screen will appear, and after you confirm that you've read and agree to the EULA, you'll see the list of components that need to be installed. The number of items may vary, depending on what components are already installed; MDAC, Internet Explorer 6.0, and FrontPage Server Extensions are just some of the possibilities. Figure 2-8 shows the list.


Visual Studio .NET Windows Component Update Setup
Microsoft® Visual Studio® .NET
Windows® Component Update
More Information
Back
Continue
Cancel

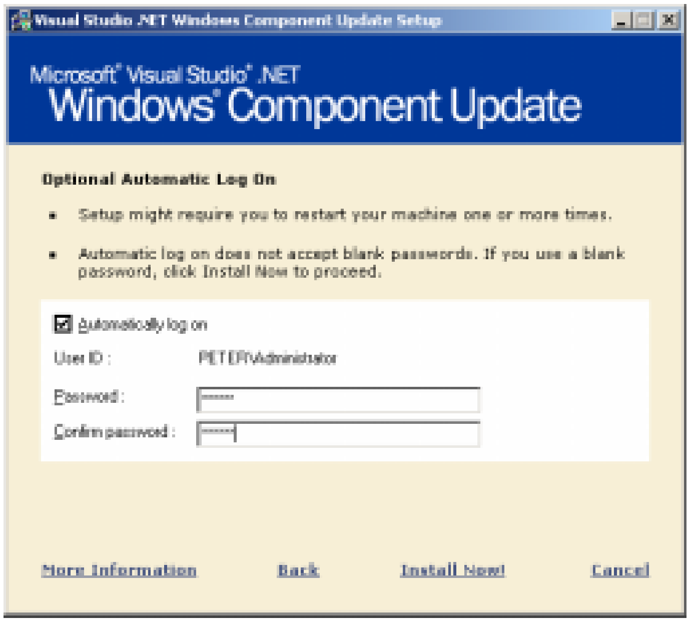
Windows Component Update will install the following components required for Visual Studio .NET

Microsoft Data Access Components 2.7
Microsoft .NET Framework

Insert 0015f0208.tif

Figure 2-8. The first step in the installation process: the Windows Component Update

Click Continue. During this part of the setup, numerous system restarts may be required. To avoid having to re–log in each time, the setup allows you to enter in your login details and then it will automatically log in for each restart during this process. This allows you to start the installation and leave it running without having to constantly check back to see whether you need to reenter your login details. This functionality is very handy, and I strongly recommend you make use of it (see Figure 2-9), as it definitely makes the Windows Component Update process less tedious.


Visual Studio .NET Windows Component Update Setup
Microsoft® Visual Studio® .NET
Windows® Component Update
More Information
Back
Install Now!
Cancel

Optional Automatic Log On

- Setup might require you to restart your machine one or more times.
- Automatic log on does not accept blank passwords. If you use a blank password, click Install Now to proceed.

☑ Automatically log on

User ID : PETER\Administrator

Password : *******

Confirm password : *******

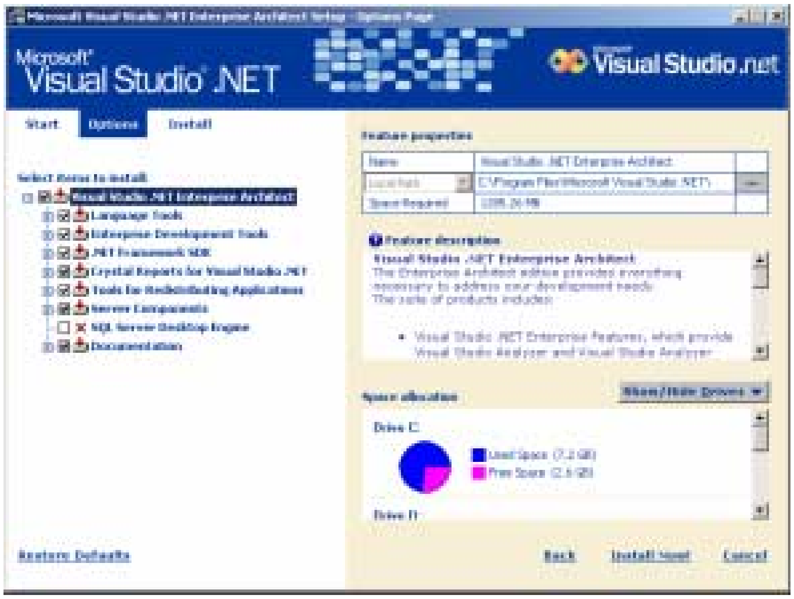
Insert 0015f0209.tif

Figure 2-9. Set up the Windows Component Update installer to automatically log in after the numerous system restarts during setup.

Once you've entered your login details, click Install Now! and the setup will proceed to install all the required components.

Once the setup has completed (it may take up to an hour, depending on the speed of your machine and the number of required components), click Done. You'll return to the original Visual Studio .NET Setup menu. Click option 2, Install Visual Studio .NET.

Once you've entered in your product key and accepted the EULA, you can click Continue. The following screen allows you to customize your VS .NET installation, as shown in Figure 2-10.



Select items to install:

- ☑ Visual Studio .NET Enterprise Architect
  - ☑ Language Tools
  - ☑ Enterprise Development Tools
  - ☑ .NET Framework SDK
  - ☑ Crystal Reports for Visual Studio .NET
  - ☑ Tools for Redistributing Applications
  - ☑ Server Components
  - □ SQL Server Desktop Engine
  - ☑ Documentation

Name	Visual Studio .NET Enterprise Architect
Local Path	C:\Program Files\Microsoft Visual Studio .NET\
Space Required	1295.26 MB

Visual Studio .NET Enterprise Architect
The Enterprise Architect edition provides everything necessary to address your development needs.
The suite of products includes:

- Visual Studio .NET Enterprise Features, which provide Visual Studio Analyzer and Visual Studio Analyzer

Drive C:
🟦 Used Space (7.2 GB)
🟪 Free Space (2.5 GB)

Drive D:

Insert 0015f0210.tif

Figure 2-10. The Visual Studio .NET IDE (and tools) setup application

The default settings should suffice, but after you're satisfied with the available options, you can proceed by clicking Install Now!

> **NOTE** For those of you interested in the Microsoft Visio and Microsoft Visual SourceSafe components, you must install these individually—you won't find an option to install them in the Options screen of the VS .NET installation.

Again, depending on the speed of your machine and the chosen configuration, VS .NET may take up to 2 hours to install. During this time, CD installations will require the CD media to be swapped periodically. Once this step of the installation is complete, you may want to use option 3 of the Visual Studio .NET Setup menu to check for service releases for VS .NET or continue to install additional components such as Visio and SourceSafe (if applicable to your version of VS .NET), but the core installation is now complete, and VS .NET is ready to work with.

Summary

IIS is an integral component of ASP.NET application development, and it's important that you familiarize yourself with the Internet Services Manager (ISM) console, particularly with regard to the Default Web Site Properties dialog box, which includes a wealth of options that govern much of how Web sites are hosted in the IIS environment. Understanding what virtual directories are and how to administer them is also very important, as VS .NET makes heavy use of them and creates a new virtual directory for every ASP.NET project that you create in VS .NET.

Beyond IIS, the installation of VS .NET and the .NET Framework (which integrates ASP.NET into IIS) is very simple, and you should be up and running without any headaches.

Chapter 3 provides an introduction to HTML and Web design techniques for the benefit of Visual Basic programmers who haven't worked with raw HTML before, as a basic understanding of how static Web sites are built is absolutely essential when programming ASP.NET applications. Topics covered include HTML, client-side JavaScript, CSS, and more, so there might even be something in there for the ASP developers among you as well.

Chapter 3

A Quick Introduction to Web Programming and VB .NET

This chapter aims to serve as both a primer for developers who have not yet had any experience in building even static (noninteractive) Web sites and an introduction to the theory behind building Web-based applications using ASP.NET and VB .NET.

HTML Primer

HTML is a standardized metalanguage used primarily on the Web for formatting documents. Web browsers are effectively HTML "interpreters." All the Web applications you develop will, once they have been processed, produce HTML for Web browsers. A significant part of your Web applications will also be raw HTML that is not produced via code. Visual Studio .NET's Web Form Designer (which I introduce in the next chapter) provides a "What You See Is What You Get" (WYSIWYG) interface for designing the user interfaces of Web applications (which is effectively what HTML is for: designing Web user interfaces). This means that you can use the Designer as you would a rich word processor, such as Microsoft Word, to write static text, format it, add images and tables, and so forth, and Visual Studio .NET will produce the HTML code behind the scenes.

However, when programming ASP.NET applications, you are really dealing with producing HTML. Having a solid understanding of what HTML is, what rules apply to it, and how it works will greatly affect your understanding of the inner workings of ASP.NET applications, which is crucial to developing solid, stable, usable, and cross-browser–compatible applications. Additionally, developing Web application user interfaces is vastly different from developing desktop application user interfaces in environments such as Visual Basic 6.0. In traditional Visual Basic, developers never really had a need to look at and understand the code that was generated behind the scenes when controls were dropped onto a form. With Web applications, this understanding is almost essential, and the code generated behind the scenes is, for the most part, HTML.

What Is HTML?

HTML, or *Hypertext Markup Language,* is a standardized language that enables documents to be formatted and linked together.

These documents are stored in ASCII text format and all formatting is in the form of *tags,* which are formatting commands (often abbreviations) placed inside the < and > characters. HTML was designed from the very beginning to be cross-platform and, as such, HTML itself contains nothing that is proprietary to any one platform. HTML is synonymous with the Internet, and in particular, the Web. This is because HTML is the standard used for transmitting documents on the Web. All Web sites are formatted using HTML (although it is possible to make files available for download—for example, Microsoft Word documents).

Possibly one of the most important characteristics of the Internet is the capability for documents to be linked to one another. It is this feature that has set the Web apart from most other data sources. For example, a news article can contain a link to a related story, such that if the user decides to click the link, the related story will display. HTML also natively provides this functionality for linking documents.

A Brief History of HTML

Tim Berners-Lee created HTML, along with the HTTP protocol, in his quest to provide a system where fellow scientists could exchange information. He came up with the concepts of HTTP, HTML, and Uniform Resource Locators (URLs, aka Web addresses), and is thus recognized as the inventor of the Web.

HTML 0 was very rudimentary and only provided the option to include a title; denote paragraphs; and insert lists, images, and links to other documents. In 1992, Dan Connolly began developing HTML 1, which attempted to address the shortcomings of HTML 0 and was released the same year. HTML 1 included several changes, but most important, it introduced the separation of the “head” of the document, which contained information about the document such as the title and the context within the Web, from the “body,” which contained the content. Another particularly pertinent introduction was that of forms, which enabled interaction between the user and the page with CGI scripts (discussed in Chapter 1).

HTML+, developed by Dave Raggett, made the inclusion of multimedia elements into documents possible and gave a certain degree of control over the formatting of text on the page. HTML 2 served to solidify HTML+ and HTML 1 and provide the industry with a definite specification. HTML 3 intended to remove formatting from HTML and include it in style sheets (I discuss style sheets a little later on in this chapter). However, before the specification could be completed, Microsoft and Netscape pushed ahead with their browsers, and the Web standards governing body, the World Wide Web Consortium (W3C), formed the HTML 3.2 specification to represent a common denominator of what had been implemented in Internet Explorer and Netscape Navigator’s 3.0 versions. The HTML 3.2 specification included the option of using embedded style sheets, as well as client-side script. Common practices, such as the definition of background, text, and link colors, as well as layout for images, were included in the HTML 3.2 specification. HTML 4, the most current specification, includes tags for completely separating style sheets from documents. It also introduces several container elements to make the integration of style sheets simpler.

> **NOTE** You may be wondering who comes up with these specifications for HTML. In 1994, upon the original request of Tim Berners-Lee, the creator of HTML, the World Wide Web Consortium (W3C) was formed. Its aim is to act as a standards body for Web-related technologies. Microsoft, Netscape, Sun Microsystems, IBM, and many other influential companies are members of the W3C, and all are allowed to make standards recommendations.

> Standardization is an extremely important issue, and although previously standards were not adhered to (particularly in the area of Web browsers and HTML), companies are becoming much more aware of the consequences of not following the standards laid out and adding their own features at will. For example, Microsoft's implementation of the SOAP and XML specifications in .NET are almost perfectly in line with those set out and agreed upon by the W3C.

Standardization and Implementation

As I've briefly touched on, HTML is one of the Web standards that the W3C regulates. Because it was the technology that the W3C was formed to regulate, the W3C had many hard lessons to learn from its experiences.

From the developer's point of view, one of the biggest challenges was that the W3C needed to keep ahead of the industry and produce final specifications before products shipped, and then try and ensure that companies complied with the rules set out by the specifications. They learned this because of the dramatic differences between the two major contenders in the browser market: Microsoft Internet Explorer and Netscape Navigator.

In version 2.0, the browsers were not radically different, as HTML was still very rudimentary. However, when the W3C took too long to develop the HTML 3 specification, both Microsoft and Netscape pushed ahead with their browser releases, and the result was chaos. Both companies had introduced client-side scripting and style sheet functionality, among a whole host of additional formatting commands. The problem was that although there were similarities between Microsoft and Netscape's implementations, the differences were vast. The W3C had to consolidate and produce HTML 3.2, which was a clear case of the W3C missing the bus, so to speak. HTML 4 was really an attempt to correct this, but it seems that Microsoft and Netscape's browsers are forever to offer slightly—and in some cases, significantly—different implementations.

The version 4.0 browsers broke away from each other significantly and developers often resorted to building two versions of a site: one for Internet Explorer and one for Netscape. However, the browsers have not evolved significantly in terms of features made available to developers since version 4.0. Microsoft Internet Explorer 5.0 offered a new method of client-side script code reuse in what is known as "behaviors," which are implemented through style sheets. However, most other advances related to speed, stability, and ease of use. Netscape skipped version 5.0, and although version 6.0 does attempt to bring Netscape closer to the functionality offered by Internet Explorer, it still has several important differences in the way it renders certain items, and some very important differences in the way that it implements client-side scripting and how it allows the developer to dynamically manipulate the document. Internet Explorer 6.0 offers little in the way of innovation for developers and, like Netscape 6.0, focuses more on the end-user experience.

HTML Document Structure, and Basic HTML Rules and Concepts

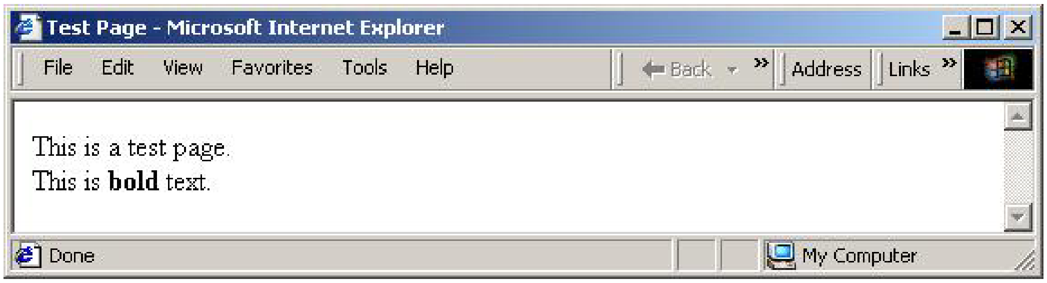
HTML documents consist primarily of HTML elements and content. HTML elements are essentially HTML tags and what they represent in a document. HTML tags are formatting commands used to determine how, and in certain instances what, content should be displayed. Tags take the form of a command inside the less-than (<) and greater-than (>) characters. To determine the amount of content that these commands apply to, most tags will have an obligatory *closing tag,* which is the same command, also inside the less-than and greater-than characters, but the less-than character is followed directly by a forward slash (/). The first tag is known as the *opening tag.* This tag will appear before the content that it applies to, and the closing tag will follow the content that is applies to, enclosing it.

Probably the best way to demonstrate HTML is to start off with an example. Listing 3-1 produces the result shown in Figure 3-1, which is a screen shot of Microsoft Internet Explorer with the HTML page loaded.

Listing 3-1.

```
<html>
<head>
 <title>Test Page</title>
</head>

<body>

 This is a test page.<br>
 This is <b>bold</b> text.<br>

</body>

</html>
```

Insert 0015f0301.tif



This is a test page.
This is **bold** text.

Figure 3-1. A simple HTML example

To try this out for yourself, perform the following steps:

1. Open up Microsoft Notepad.
2. Type out the code in Listing 3-1.
3. Save the file using a .html extension rather than a normal .txt extension (make sure you select All Files from the “Save as type” combo box in the Save As dialog box to avoid the .html extension being taken as part of the filename and .txt being appended).

4. Open up Microsoft Internet Explorer and navigate to the full path of the file using the address bar. For example, if you saved the file to C:\somefolder\listing3-1.html, type that path in.

The browser should now display what is shown in Figure 3-1. This is a very simple example of an HTML page; however, it shows the basic structure that all HTML documents take, no matter how small or large they are. The entire document is encased in the <html> tag. The two primary sections in the document are the head and the body. The head section typically contains nonvisual information about the document. In this example, it only contains the title of the document, which you will notice is reflected in the title bar of the Internet Explorer window. This title may also be used for indexing and search relevance purposes by search engines.

The body of the document contains, as the name suggests, the content of the document. It can contain plain text only, although even in using only plain text, one very important aspect of HTML is evident: Line breaks need to be explicitly declared. The <br>, or "break," tag is used to produce line breaks. Each tag produces one line break, so if you wanted to produce two line breaks, you'd use two <br> tags in succession, like this:

```
Some text...<br><br>
```

Another important concept in HTML is the handling of white space. HTML only produces one space at a time, unless a special character entity is used. The first line in the code that follows produces exactly the same output as the second line:

```
This     is    some          text.

This is some text.
```

The last important concept demonstrated in Listing 3-1 is that of formatting. In this particular instance, only one word is formatted. The example used the <b>, or "bold," tag to add bold formatting to the text within the opening and closing tags. Naturally, you can bold more than one word at a time by enclosing all the words to be formatted inside the opening and closing tags, as follows:

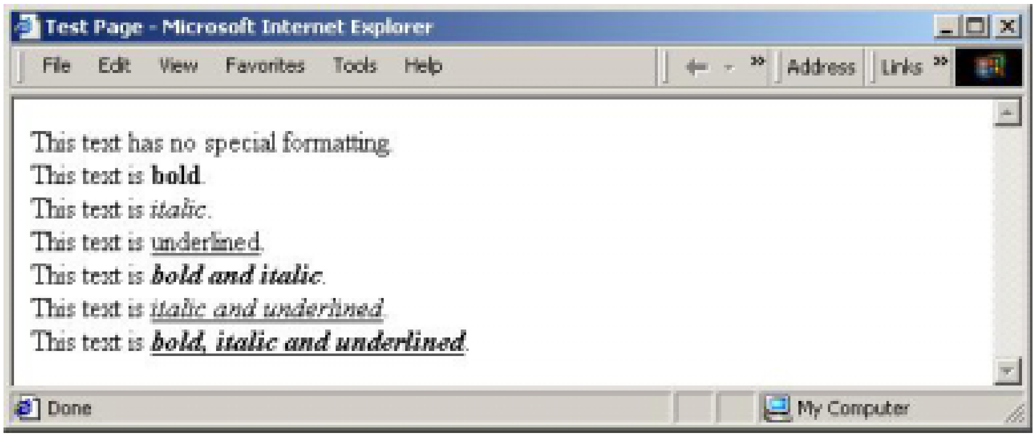
```
This text is not bold. <b>All this text is bold.</b> This text is not bold.
```

Several other formatting tags are commonly used. The next example introduces two more, <u> for underlining and <i> for italicizing text, to demonstrate several other HTML concepts. Listing 3-2 demonstrates the combined use of the <b>, <u>, and <i> tags.

Listing 3-2.

```
<html>
<head>
 <title>Test Page</title>
</head>

<body>

 This text has no special formatting.<br>
 This text is <b>bold</b>.<br>
 This text is <i>italic</i>.<br>
 This text is <u>underlined</u>.<br>
 This text is <b><i>bold and italic</i></b>.<br>
 This text is <i><u>italic and underlined</u></i>.<br>
 This text is <b><i><u>bold, italic and underlined</u></i></b>.<br>

</body>

</html>
```

Figure 3-2 shows the output of Listing 3-2 in Internet Explorer.


Test Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

This text has no special formatting.
This text is **bold**.
This text is *italic*.
This text is underlined.
This text is ***bold and italic***.
This text is *italic and underlined*.
This text is ***bold, italic and underlined***.

Insert 0015f0302.tif

Figure 3-2. Simple HTML formatting

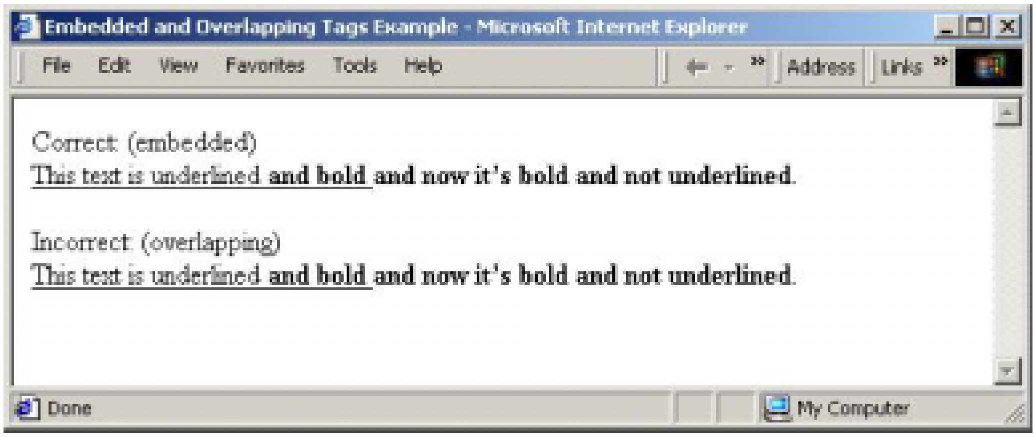
This example shows how HTML tags can be embedded within each other. There is one simple rule that you need to abide by: Embedded tags must be embedded, not overlapped. Listing 3-3 demonstrates both embedded and overlapping tags.

Listing 3-3.

```
<html>
<head>
 <title>Embedded and Overlapping Tags Example</title>
</head>

<body>

 Correct: (embedded)<br>
 <u>This text is underlined <b>and bold</b> </u><b>and now it's bold
and
  not underlined</b>.<br><br>

 Incorrect: (overlapping)<br>
 <u>This text is underlined <b>and bold </u>and now it's bold and not
  underlined</b>.
```

```
</body>

</html>
```

Both examples will presumably produce the output shown in Figure 3-3.


Embedded and Overlapping Tags Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Links
Done
My Computer

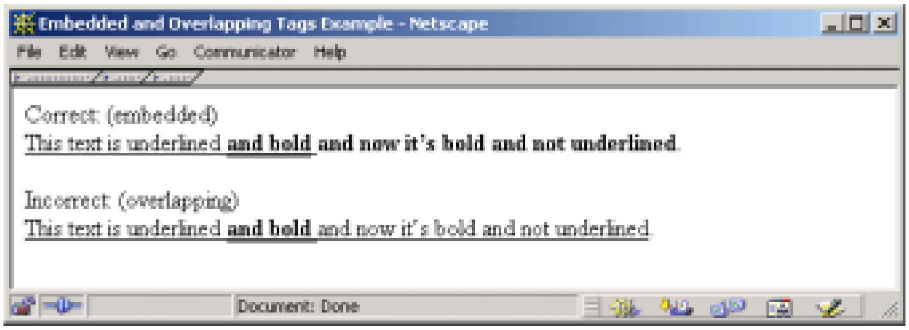
Correct: (embedded)
This text is underlined and bold and now it's bold and not underlined.

Incorrect: (overlapping)
This text is underlined and bold and now it's bold and not underlined.

Insert 0015f0303.tif

Figure 3-3. Overlapping tags render as expected in Internet Explorer

The problem with overlapping tags is that they produce unpredictable output. In several versions of Netscape's browser, Listing 3-3 will produce the output displayed in Figure 3-4.


Embedded and Overlapping Tags Example - Netscape
File Edit View Go Communicator Help
Document: Done

Correct: (embedded)
This text is underlined and bold and now it's bold and not underlined.

Incorrect: (overlapping)
This text is underlined and bold and now it's bold and not underlined.

Insert 0015f0304.tif

Figure 3-4. Overlapping tags don't render as expected in Netscape Navigator

The basic lesson to be learned here is never using overlapping tags. Being lazy and not simply adding in an additional opening and closing tag will probably result in unpredictable output and wasted time, so adhere to the rule and you'll save yourself a lot of time and effort.

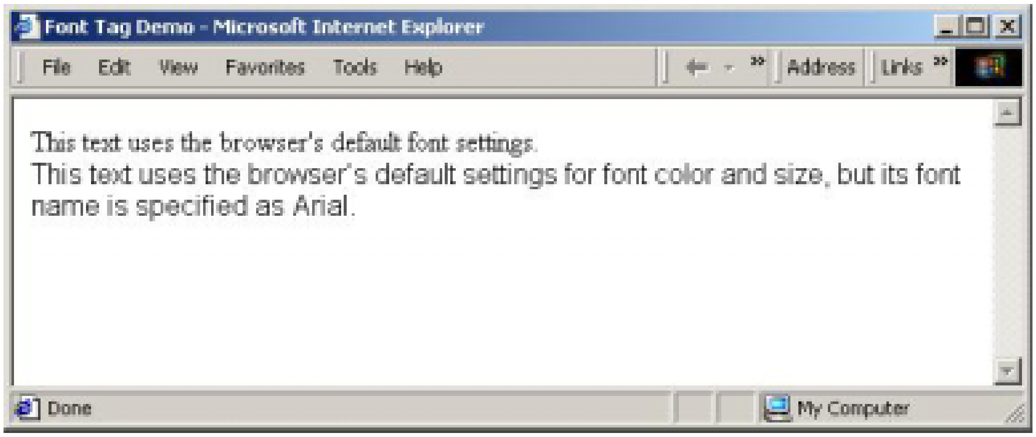
Formatting HTML Documents

The three formatting tags that you have learned about thus far offer very rudimentary functionality. The <font> tag allows for the definition of the font color, size, and name for text enclosed between the opening and closing tags. The <font> tag introduces the concept of HTML attributes. In the previous three formatting tags that have been used, their value is either on or off—the text is either bold or not bold. There is no degree of boldness, so to speak. With font color, size, and name, there is no “on or off” scenario. You need to make an explicit definition of the font color, size and/or name. For this purpose, HTML provides attributes. *Attributes* allow you to supply additional information with a tag. Listing 3-4 shows the use of the font tag specifying the font name using the face attribute.

Listing 3-4.

```
<html>
<head>
 <title>Font Tag Demo</title>
</head>

<body>

 This text uses the browser's default font settings.<br>
 <font face="Arial">This text uses the browser's default settings for
font color and size, but its font name is specified as Arial.</font>

</body>

</html>
```

Figure 3-5 shows how Listing 3-4 looks in Internet Explorer.



This text uses the browser's default font settings.
This text uses the browser's default settings for font color and size, but its font name is specified as Arial.

Insert 0015f0305.tif

Figure 3-5. Formatting text using different fonts

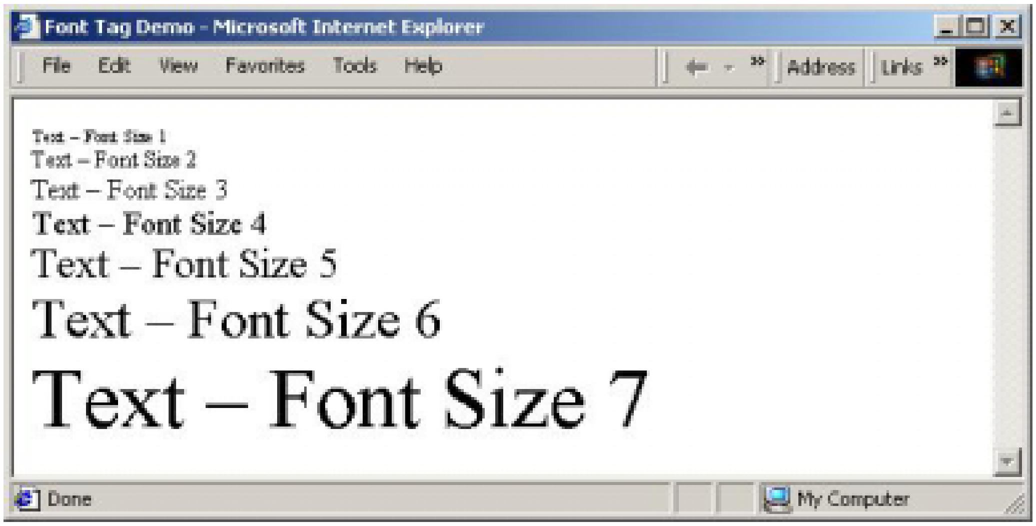
The face attribute allows the font name to be specified. Any font that is installed on the viewer’s machine can be specified and will be used. This has a fairly significant implication: Because the fonts on the client’s machine are the only ones that can be used, you should only specify common fonts (i.e., fonts installed by Windows by default) so that your HTML pages have a consistent look on most, if not all, machines.

The size attribute does not work in the conventional sense that it uses the points (pt) or pixels (px) measurements for font size. It uses its own scale of between 1 and 7, with 1 being the smallest size and 7 being the largest size. Listing 3-5 demonstrates the use of the size attribute.

Listing 3-5.

```
<html>
<head>
 <title>Font Tag Demo</title>
</head>

<body>

 <font size="1">Text – Font Size 1</font><br>
 <font size="2">Text – Font Size 2</font><br>
 <font size="3">Text – Font Size 3</font><br>
 <font size="4">Text – Font Size 4</font><br>
 <font size="5">Text – Font Size 5</font><br>
 <font size="6">Text – Font Size 6</font><br>
 <font size="7">Text – Font Size 7</font><br>

</body>

</html>
```

Figure 3-6 shows the output of Listing 3-5 in Internet Explorer.



Text – Font Size 1
Text – Font Size 2
Text – Font Size 3
Text – Font Size 4
Text – Font Size 5
Text – Font Size 6
Text – Font Size 7

Insert 0015f0306.tif

Figure 3-6. Formatting text using font sizes

Size 3 is the standardized default setting for the rendition of text in HTML documents in browsers.

The color attribute accepts two types of values: color constants and color values (in the form RGB). *Color constants* are plain English names for colors, such as “Black,” “White,” “Red,” “Green,” “Blue,” and so forth. A complete list is available in the appendixes. The use of color constants can lead to problems, however. This is because different browsers support different color constants. For example, Internet Explorer 3.0 and over has support for “NavyBlue,” which colors text light blue. However, most other Internet browsers do not support this constant, and thus text remains black. The list of color constants included in HTML 4 specification, and generally supported by browsers, is far from comprehensive. Thus, to have any precision and proper control over the colors used, specific color values must be used for the color attribute.

Color values consist of the three parts: the amount of red to be used, the amount of green to be used, and the amount of blue to be used. These “amounts” are on a scale of 0 to 255, with 0 being the least amount of the color used and 255 being the most. For example, fuchsia is an equal mixture of red and blue. The value would then be 255; 0; 255. However, in HTML, *hex values* are used to represent colors, and thus all values must be converted from decimal to hexadecimal. *Hexadecimal* is a method of representing data that uses 16 “digits”—0 to 9, and then A to F—and is for this reason known as *base 16.* In hex, 255 is FF and 0 is 0. The value for fuchsia is FF00FF. Note that an extra zero has been added to pad the green value, which would only have been one figure. A hash mark (#) is also inserted at the beginning of the string to indicate that a color value is being used, and not a color constant, so the final color value for fuchsia is #FF00FF.

Color values can be used to obtain more than 16 million different color combinations. This brings with it the problem of ensuring that colors look the same, or as close to identical as possible, on all machines. A so-called Web-safe palette has been devised in which all colors in it should look identical on most machines. The palette follows a simple rule: The intensities of colors in the RGB string must be 00, 33, 66, 99, CC, or FF. So #FFCC00 (orange) is a Web-safe color, whereas #FEAB05 (also orange) is not. The Web-safe palette contains only 216 colors (6×6×6 combinations compared to 256×256×256), but the colors are much more likely to appear the same on all machines than the more precise ones using the full possible palette available are.

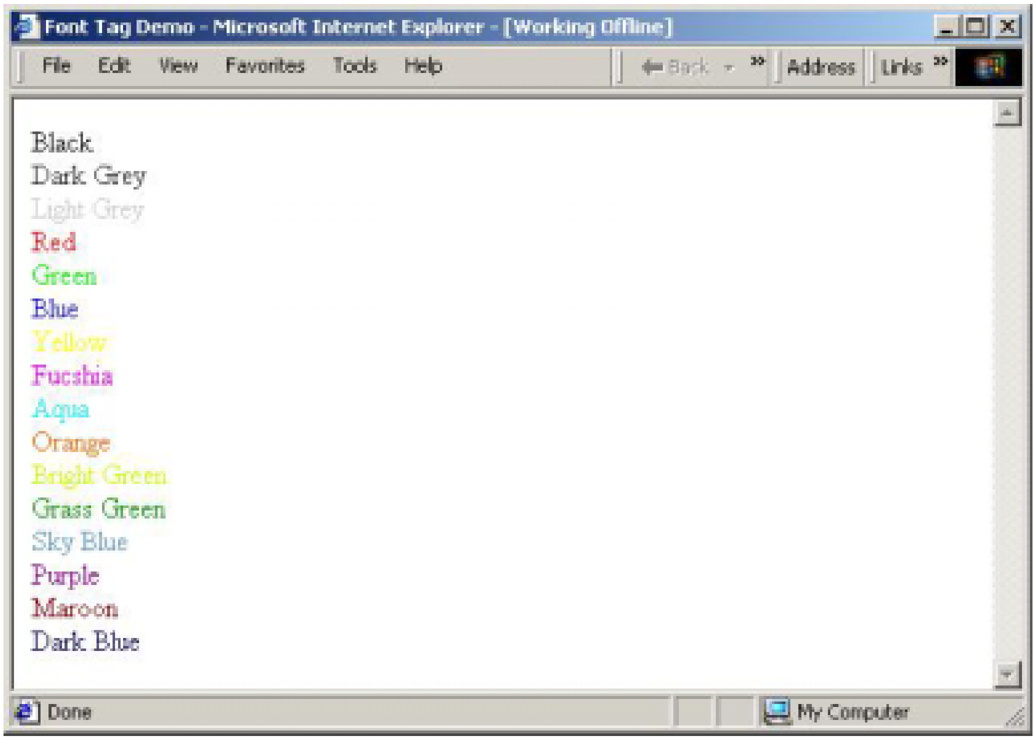
Listing 3-6 demonstrates the use of colors with the font tag.

Listing 3-6.

```
<html>
<head>
 <title>Font Tag Demo</title>
</head>

<body>

 <font color="#000000">Black</font><br>
 <font color="#333333">Dark Grey</font><br>
 <font color="#CCCCCC">Light Grey</font><br>
 <font color="#FF0000">Red</font><br>
```

```
  <font color="#00FF00">Green</font><br>
  <font color="#0000FF">Blue</font><br>
  <font color="#FFFF00">Yellow</font><br>
  <font color="#FF00FF">Fuchsia</font><br>
  <font color="#00FFFF">Aqua</font><br>
  <font color="#FF6600">Orange</font><br>
  <font color="#CCFF00">Bright Green</font><br>
  <font color="#009900">Grass Green</font><br>
  <font color="#6699CC">Sky Blue</font><br>
  <font color="#990099">Purple</font><br>
  <font color="#990000">Maroon</font><br>
  <font color="#000066">Dark Blue</font><br>

</body>

</html>
```

Figure 3-7 shows the output of Listing 3-6 in Internet Explorer.


Font Tag Demo - Microsoft Internet Explorer - [Working Offline]
File
Edit
View
Favorites
Tools
Help
Back
Address
Links
Done
My Computer

```
Black
Dark Grey
Light Grey
Red
Green
Blue
Yellow
Fucshia
Aqua
Orange
Bright Green
Grass Green
Sky Blue
Purple
Maroon
Dark Blue
```

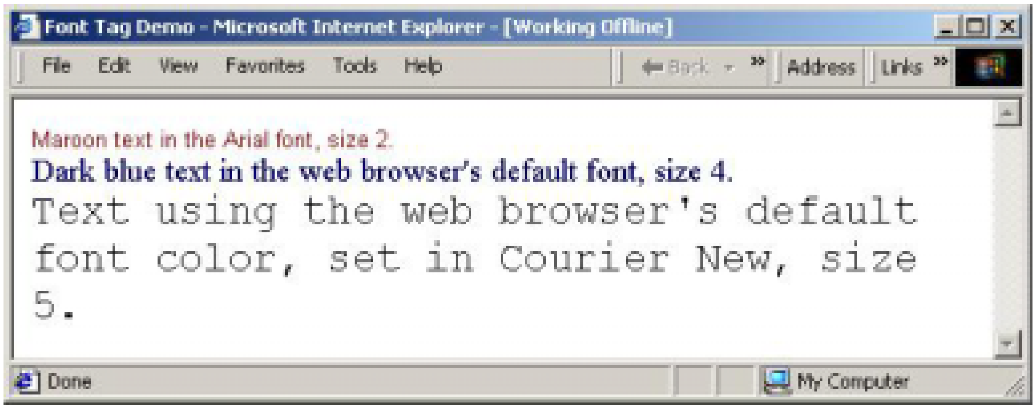
Insert 0015f0307.tif

Figure 3-7. Formatting text using color

You can use more than one attribute at a time to allow multiple simultaneous formatting options with the font tag. Listing 3-7 shows how font size, font color, and font face can be used in conjunction with each other, as well as in smaller combinations. Figure 3-8 shows the output.

Listing 3-7.

```
<html>
<head>
 <title>Font Tag Demo</title>
</head>

<body>

 <font color="#990000" face="Arial" size="2">Maroon text in the
Arial font, size 2.</font><br>
 <font color="#000066" size="4">Dark blue text in the web browser's
default font, size 4.</font><br>
 <font face="Courier New" size="5">Text using the web browser's
default font color, set in Courier New, size 5.</font><br>

</body>

</html>
```


Font Tag Demo - Microsoft Internet Explorer - [Working Offline]
File
Edit
View
Favorites
Tools
Help
Back
Address
Links
Done
My Computer

Maroon text in the Arial font, size 2.
Dark blue text in the web browser's default font, size 4.
Text using the web browser's default font color, set in Courier New, size 5.

Insert 0015f0308.tif

Figure 3-8. Formatting text using different font faces, font sizes, and font colors

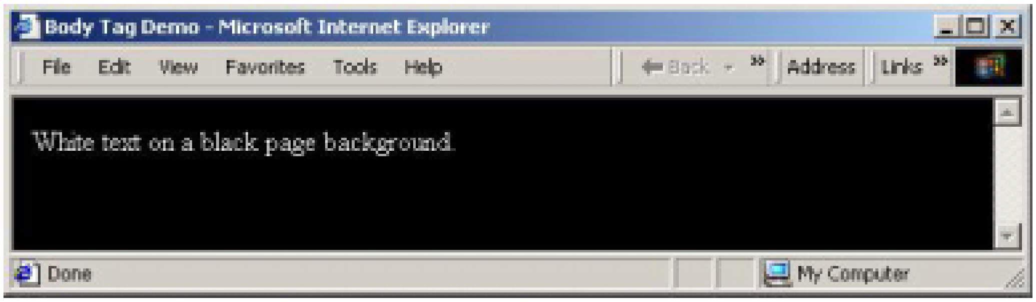
Numerous other tags deal with the formatting of text; however, those omitted replicate the functionality of the tags already introduced in this chapter, and they are used normally to more accurately describe the reason for formatting a particular piece of text. For example, the address tag generally produces the same result as using the italic (i) tag.

All HTML documents contain a body tag. This tag is not only used to define the bounds of the content in the document, but it also features numerous attributes for setting default values and changing the way the document looks. The pertinent attributes for the moment are bgcolor, background, text, link, vlink, and alink.

The bgcolor attribute specifies, as its name suggests, the background color for the document, which is by default white (in most browsers). The values that this attribute accepts are identical to those of the font tag's color attribute: color constants or hexadecimal color values. Listing 3-8 shows the use of the bgcolor attribute and Figure 3-9 shows the output of Listing 3-8.

Listing 3-8.

```
<html>
<head>
 <title>Body Tag Demo</title>
</head>

<body bgcolor="#000000">

 <font color="#FFFFFF">White text on a black page
background.</font><br>

</body>

</html>
```



White text on a black page background.

Insert 0015f0309.tif

Figure 3-9. Modifying the page's background color

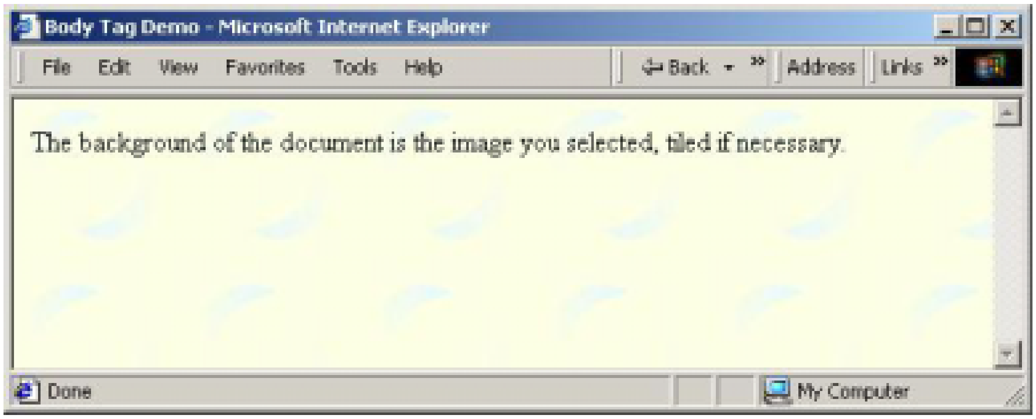
The background attribute allows for an image to be tiled in the background of the page. The two most common formats for images used on the Internet are Joint Photographic Experts Group (JPEG) and Graphics Interchange Format (GIF), although a third, Portable Network Graphics (PNG), is generally also supported by Web browsers. To try out using the background attribute, do the following:

1. Type out Listing 3-9 and save the file as bgtest.html.
2. Copy/create a JPEG image to the same directory in which you saved the HTML file and rename it bg.jpg. The image should preferably have dimensions of less than 100×100 so tiling will be demonstrated, and it should also be fairly light so that there will be a contrast between the text and the background.
3. Load bgtest.html into Internet Explorer. The output should (depending on the image used) look similar to Figure 3-10.

Listing 3-9.

```
<html>
```

```
<head>
 <title>Body Tag Demo</title>
</head>

<body background="bg.jpg">

 The background of the document is the image you selected, tiled if necessary.

</body>

</html>
```



The background of the document is the image you selected, tiled if necessary.

Insert 0015f0310.tif

Figure 3-10. Setting an image as the page background

Begin sidebar

Understanding File Paths in HTML

There are essentially two different ways in which you can reference external documents in HTML: absolute and relative file paths.

Absolute paths use the entire URL of a file, such as http://www.dotnet.za.net/images/top_title.jpg. The use of absolute paths should be avoided if at all possible because they restrict portability. For example, if a site using absolute paths is moved to a different location, all the paths in all the files need to be changed to reflect the new location.

Relative paths, however, do not include the entire URL of a file. Relative paths provide the location of the file in relation to the file referencing it. For example, assume there is an HTML file located at http://www.dotnet.za.net/somefile.html. This page is required to use a background picture located at http://www.dotnet.za.net/images/bg.jpg. The page would reference the image by using images/bg.jpg rather than including the full URL. If the image was located at http://www.dotnet.za.net/images/backgrounds/bg.jpg, the page would reference the image by using images/backgrounds/bg.jpg.

Relative paths easily reference files not only in higher levels, but also in lower directory structure levels with the level down string (..). Assuming that an image is located at http://www.dotnet.za.net/bg.jpg and a page is located at http://www.dotnet.za.net/anydir/somepage.html, the page would reference the image in its parent directory by using ../bg.jpg. If the page was located at http://www.dotnet.za.net/anydir/anotherdir/somepage.html, it would reference using ../../bg.jpg.

The final type of relative path is where neither of the files is in different directories. In this case, the path is simply the filename, as demonstrated in Listing 3-9. The last important item to remember is that all paths in HTML take the same format, whether they are absolute or relative.

End sidebar

TIP If a background image is being used, it is useful to include the bgcolor attribute in addition to the background attribute with a color value as close to the dominant color or shade of the image. Should the image take a while to load on slow connections, a contrast will still be maintained so that the text on the page can be read.

The text, link, vlink, and alink attributes of the body tag all take color values. The text attribute is used to define the default font color for all text on the page (naturally, this can be overridden using the font tag). The link attribute is used to define the color of unfollowed links (links that have not been clicked) on the page; vlink defines links that have been followed; and alink defines links that have been clicked, but the file that they link to has not started loading.

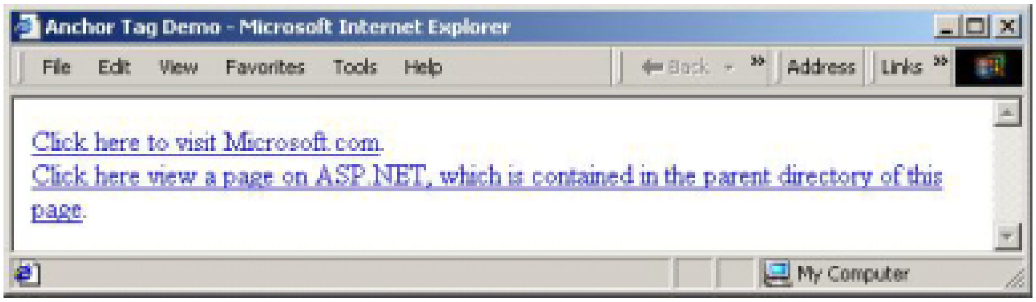
Linking Pages

One of the major advantages of the World Wide Web is the capability of pages to be linked to each other. Links are created using the a, or "anchor," tag, in conjunction with the href attribute, which contains the path (absolute or relative) of the file to link to. Listing 3-10 demonstrates the use of the anchor tag to create links. Figure 3-11 shows the output.

Listing 3-10.

```
<html>
<head>
 <title>Anchor Tag Demo</title>
</head>

<body>

 <a href="http://www.microsoft.com">Click here to visit
Microsoft.com</a>.<br>
```

```
  <a href="../aspdotnet.html">Click here to view a page on ASP.NET, which
 is contained in the parent directory of this page</a>.<br>

</body>

</html>
```



Click here to visit Microsoft.com.
Click here view a page on ASP.NET, which is contained in the parent directory of this page.

Insert 0015f0311.tif

Figure 3-11. Links using absolute and relative paths

> **NOTE** The file that is being linked to does not necessarily have to be an HTML document. You can create links to ZIP files, Microsoft Word documents, and so forth. Modern Web browsers will display a dialog box to allow the user to save the file to his or her hard drive. This is because browsers detect that the content being sent is not HTML. This is how *file downloads* are created—they are simply links to files other than HTML documents.

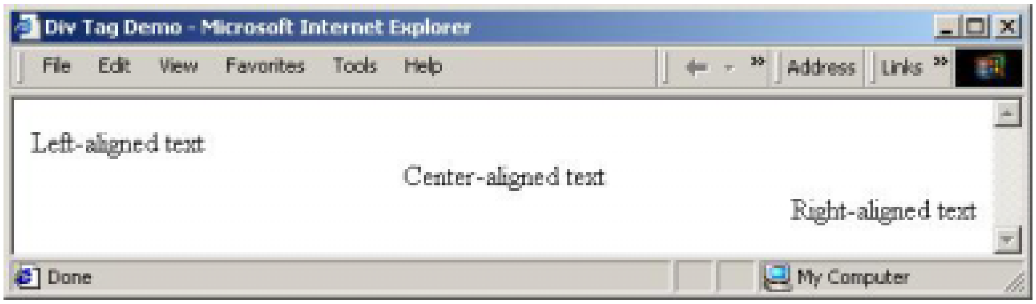
Regions

It is possible to define regions of content in an HTML page. For now, the primary purpose of this is for alignment, although there are several other important reasons to define regions. The two tags for defining regions are div and span. Put simply, the difference between the two is that div takes up the entire width of the page and span is only the width of the content in it. It is for this reason that span elements are more typically used for formatting through style sheets than for alignment.

The div tag has an align attribute, which accepts left, center, or right. Listing 3-11 uses all three. Figure 3-12 shows the output.

Listing 3-11.

```
<html>
<head>
  <title>Div Tag Demo</title>
</head>

<body>

  <div align="left">Left-aligned text</div>
  <div align="center">Center-aligned text</div>
  <div align="right">Right-aligned text</div>

</body>

</html>
```


Div Tag Demo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Links
Done
My Computer

Left-aligned text

Center-aligned text

Right-aligned text

Insert 0015f0312.tif

Figure 3-12. Alignment using divisions

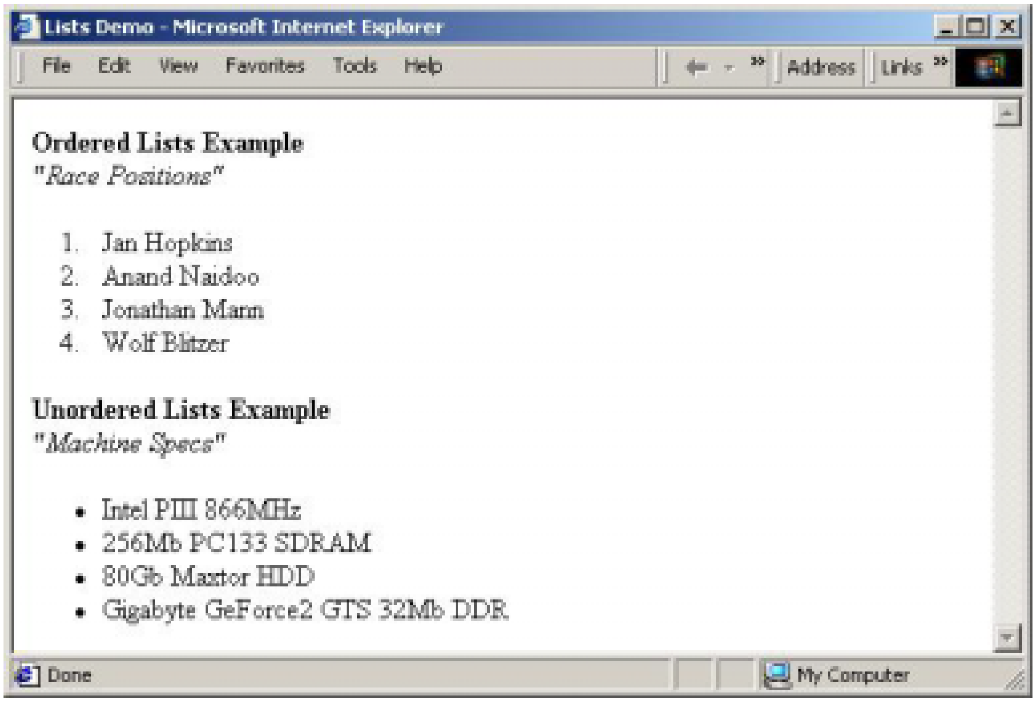
Again, note that each div takes up the entire width of the page, so each div is effectively on a new line. Divs can contain multiple lines of text and text-related tags, such as font, as well as images, tables, lists, and even other divs.

Lists

HTML provides two sets of tags for creating lists: ol for “ordered lists” and ul for “unordered lists”; as well as the li for “list item” tag for adding items. The advantage of using lists in HTML is that alignment and numbering (in the case of ordered lists) is done automatically. Listing 3-12 demonstrates the use of both types of lists, and Figure 3-13 shows the output.

Listing 3-12.

```
<html>
<head>
  <title>Lists Demo</title>
</head>

<body>

  <b>Ordered List Example</b><br>
  <i>"Race Positions"</i><br><br>
  <ol>
    <li>Jan Hopkins</li><br>
    <li>Anand Naidoo</li><br>
    <li>Jonathan Mann</li><br>
    <li>Wolf Blitzer</li><br>
  </ol>
  <b>Unordered List Example</b><br>
  <i>"Machine Specs"</i><br><br>
  <ul>
   <li>Intel PIII 866MHz</li><br>
   <li>256Mb PC133 SDRAM</li><br>
   <li>80Gb Maxtor HDD</li><br>
   <li>Gigabyte GeForce2 GTS 32Mb DDR</li><br>
  </ul>

</body>

</html>
```



Ordered Lists Example
"Race Positions"

1. Jan Hopkins
2. Anand Naidoo
3. Jonathan Mann
4. Wolf Blitzer

Unordered Lists Example
"Machine Specs"

- Intel PIII 866MHz
- 256Mb PC133 SDRAM
- 80Gb Maxtor HDD
- Gigabyte GeForce2 GTS 32Mb DDR

Insert 0015f0313.tif

Figure 3-13. Ordered and unordered lists

Notice that all the text is inline, as if tabs had been used (tabs are counted as white space in HTML, and thus only render as one space, regardless of how many are used). This autolineup becomes very useful when there are ten or more items in the list.

Images

You can insert images into a page by using the img tag. This tag has several important attributes:

* The src attribute is the source of the image—the path to the image file.

* The width and height attributes define the dimensions of the image. If they differ from the actual dimensions of the image, the image is distorted accordingly.

* The alt attribute allows for a short description that will be displayed until the image has started loading.

* The align attribute allows for specifying alignment, which can override the settings given to it if it is in a division with alignment settings.

Like the break tag, the image tag does not have a closing tag. Listing 3-13 shows how to insert an image with alignment specified against that of a division, and Figure 3-14 shows the result.

Listing 3-13.

```
<html>
<head>
 <title>Image and Alignment Demo</title>
</head>

<body>

 <div align="right">
  <img src="fire.jpg" width="151" height="113" alt="Fire" align="left">
  This is a picture of a fire.
 </div>

</body>

</html>
```

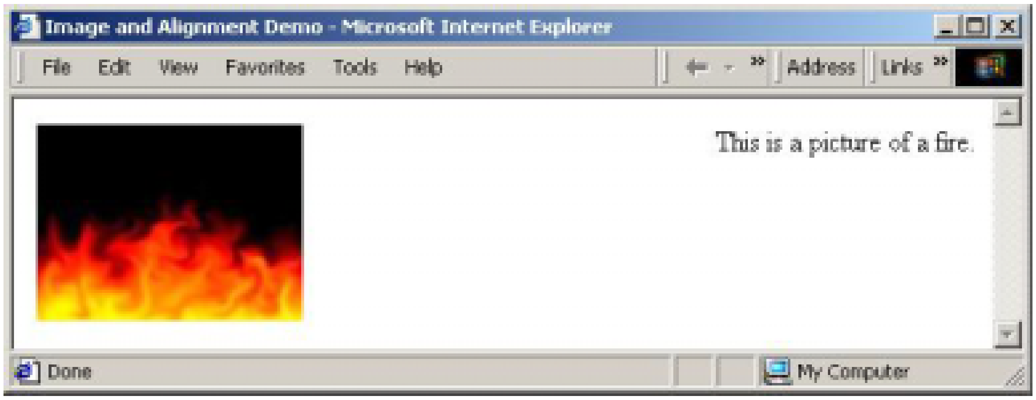


Image and Alignment Demo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
Done
My Computer

This is a picture of a fire.

Insert 0015f0314.tif

Figure 3-14. Inserting images and defining alignment

Note that the text is right-aligned; however, because the image overrode the alignment setting, it is left-aligned. If the align attribute of the image tag was omitted, the image would be right-aligned as well.

Tables

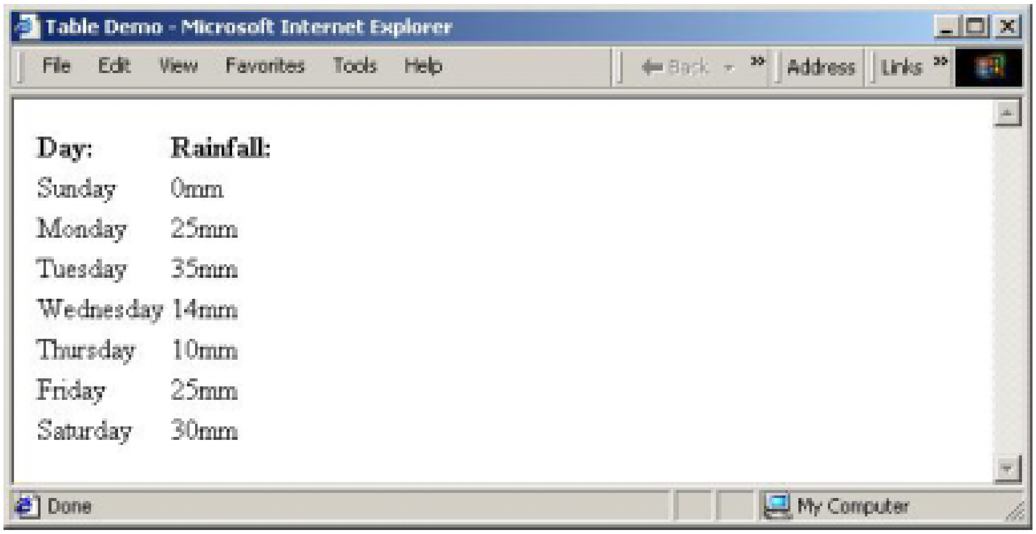
Tables are an extremely important aspect of HTML. The purpose of a table is normally to represent data. In HTML, however, tables play a much larger and more important role in page layout.

Tables consist of rows and columns. In HTML, a table is defined by the table tag; rows are defined with the tr, or “table row,” tag; and data (and hence columns) is defined with the td, or “table data,” tag. Listing 3-14 shows how a simple table is created without using any special attributes, and Figure 3-15 shows the output.

Listing 3-14.

```
<html>
<head>
 <title>Table Demo</title>
</head>

<body>

 <table>
  <tr>
   <td>Day:</td>
   <td>Rainfall:</td>
  </tr>
  <tr>
   <td>Sunday</td>
   <td>0mm</td>
  </tr>
  <tr>
   <td>Monday</td>
   <td>25mm</td>
  </tr>
  <tr>
   <td>Tuesday</td>
   <td>35mm</td>
  </tr>
  <tr>
   <td>Wednesday</td>
   <td>14mm</td>
  </tr>
  <tr>
```

```
      <td>Thursday</td>
      <td>10mm</td>
    </tr>
    <tr>
      <td>Friday</td>
      <td>25mm</td>
    </tr>
    <tr>
      <td>Saturday</td>
      <td>30mm</td>
    </tr>
  </table>

</body>

</html>
```



Day:	Rainfall:
Sunday	0mm
Monday	25mm
Tuesday	35mm
Wednesday	14mm
Thursday	10mm
Friday	25mm
Saturday	30mm

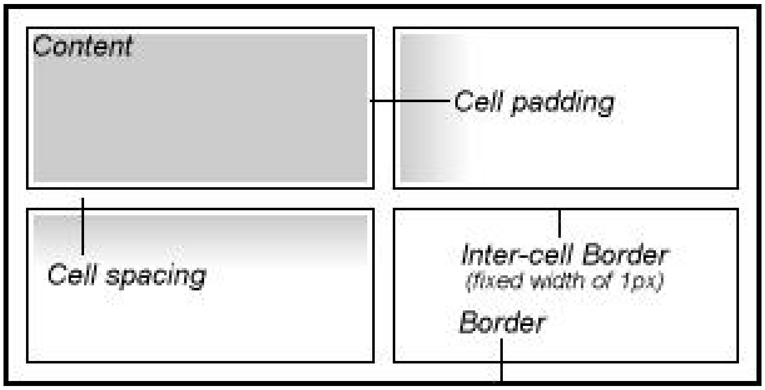
Insert 0015f0315.tif

Figure 3-15. A simple table

There are two important aspects to observe about the default way in which tables are rendered. First, there is spacing between the cells. Second, no border is included. To add a border, use the border attribute of the table tag, which accepts a numeric value from 0 (no border) upward.

> **NOTE** To exclude certain cells from having a border when the table's border attribute is set to include a border, you need to use style sheets. Style sheets are covered later in this chapter.

The table tag also has two important attributes that deal with spacing: cellpadding and cellspacing. The cellpadding attribute defines the amount of space between the content of a cell and its borders. The cellspacing attribute defines the amount of space between the cell borders. Figure 3-16 shows the relationship between the table border, the intracell borders, the cell content, the cell padding, and the cell spacing.



Insert 0015f0316scrap.tif FPO

Figure 3-16. HTML table attribute meanings

At first glance, there is seemingly little point to the separation of the cellpadding and cellspacing attributes, as both contribute to the distance between the content of the cell and the content of other cells. The difference is fairly subtle, but important. For example, let’s assume that a particular table has a cell padding of 0 and a cell spacing of 5. At first, you might think that a cell will render exactly the same if the cell padding was 5 and the cell spacing was 0, which is partially correct—the distance between the content of cells will be identical. However, in the first instance, the content will be right up against the intracell border and there will be 5 pixels of space between the intracell borders. In the second instance, there will be 5 pixels of space between the content of each cell and the intracell borders, and no space between the intracell borders.

Listing 3-15 shows the use of the table’s border and bgcolor attributes, and demonstrates the difference between cell padding and cell spacing. Figure 3-17 shows the output of Listing 3-15.

Listing 3-15.

```
<html>
<head>
 <title>Table Borders, Backgrounds and Cell Padding and Spacing
Demo</title>
</head>

<body>

 Border=3; Background color=#CCCCCC; Cellpadding=0;
Cellspacing=5<br>
 <table border="3" bgcolor="#CCCCCC" cellpadding="0" cellspacing="5">
  <tr>
```

```
      <td>Cell 1</td>
      <td>Cell 2</td>
    </tr>
    <tr>
      <td>Cell 3</td>
      <td>Cell 4</td>
    </tr>
  </table><br>
  Border=3; Background color=#CCCCCC; Cellpadding=5;
Cellspacing=0<br>
  <table border="3" bgcolor="#CCCCCC" cellpadding="5" cellspacing="0">
    <tr>
      <td>Cell 1</td>
      <td>Cell 2</td>
    </tr>
    <tr>
      <td>Cell 3</td>
      <td>Cell 4</td>
    </tr>
  </table>

</body>

</html>
```

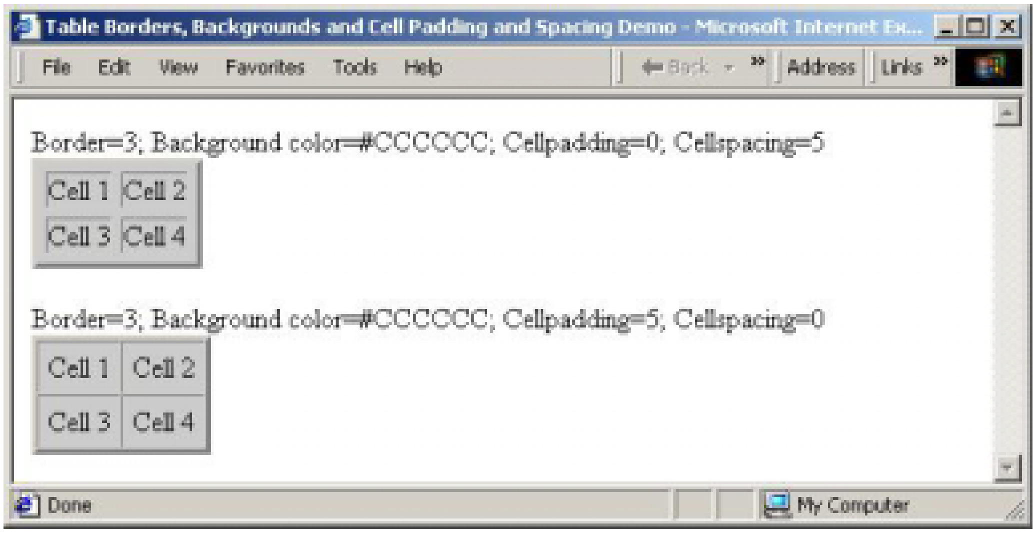


Table Borders, Backgrounds and Cell Padding and Spacing Demo - Microsoft Internet Ex...
File
Edit
View
Favorites
Tools
Help
Back
Address
Links
Done
My Computer

Border=3; Background color=#CCCCCC; Cellpadding=0; Cellspacing=5

Cell 1	Cell 2
Cell 3	Cell 4

Border=3; Background color=#CCCCCC; Cellpadding=5; Cellspacing=0

Cell 1	Cell 2
Cell 3	Cell 4

Insert 0015f0317.tif

Figure 3-17. The difference between cell padding and cell spacing

As you can see in Figure 3-17, there is quite a significant difference between cell padding and cell spacing. Certain circumstances require only cell padding to be used, some require only cell spacing, and others require both.

The only three other particularly important attributes of the table tag are the align, width, and height attributes. The align attribute specifies the alignment of the table, either left, center, or right. The width and height attributes take either pixel values or a percentage of the available space. The table data tag contains several attributes that are important to know: the bgcolor, or “background color,” attribute; the align attribute; and the valign, or “vertical align,” attribute. There are also two dimension attributes: width and height. The table row tag also contains these six attributes, and if set in a table row, they are applied to all cells in that row. Listing 3-16 demonstrates the use of most of these attributes, and Figure 3-18 shows the output of Listing 3-16.

Listing 3-16.

```
<html>
<head>
 <title>Table Demo</title>
</head>

<body>

 <table align="center" border="1" width="50%" height="250">
  <tr height="30" align="right" valign="Top">
   <td width="75%">Cell 1</td>
   <td>Cell 2</td>
  </tr>
  <tr>
   <td width="75%" height="80" align="Center" valign="Bottom">Cell
3</td>
   <td height="80">Cell 4</td>
  </tr>
 </table>

</body>

</html>
```

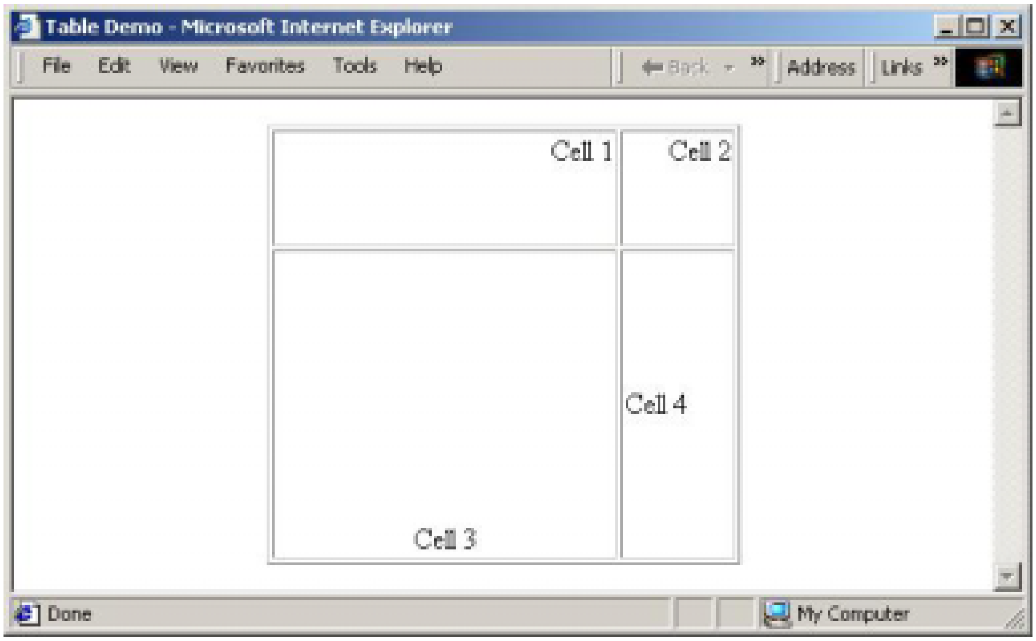


Table Demo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
Links
Done
My Computer

Cell 1	Cell 2
Cell 3	Cell 4

Insert 0015f0318.tif

Figure 3-18. Cell content alignment

As you can see in Figure 3-18, tables offer a high level of control over the layout of content. It is for this reason that tables are often central to the design and layout of pages. However, I need to introduce one more aspect of tables before you can fully utilize them. A common feature in word processing and other applications that include support for tables is the capability to merge cells. HTML provides for this with the colspan and rowspan attributes of the table data tag. Both of these attributes take the number of cells to be joined, with their default being 1. Listing 3-17 and Figure 3-19 show how these two attributes are used.

Listing 3-17.

```
<html>
<head>
 <title>Table Cell Spanning Demo</title>
</head>

<body>

 Column Span:<br>
 <table border="1">
  <tr>
   <td colspan="2">Cell 1</td>
   <td>Cell 2</td>
  </tr>
  <tr>
   <td>Cell 3</td>
```

```
      <td>Cell 4</td>
      <td>Cell 5</td>
    </tr>
  </table><br>
  Row Span:<br>
  <table border="1">
    <tr>
      <td rowspan="2">Cell 1</td>
      <td>Cell 2</td>
      <td>Cell 3</td>
    </tr>
    <tr>
      <td>Cell 4</td>
      <td>Cell 5</td>
    </tr>
  </table><br>
  Column Span and Row Span:<br>
  <table border="1">
    <tr>
      <td rowspan="2">Cell 1</td>
      <td colspan="2">Cell 2</td>
    </tr>
    <tr>
      <td>Cell 3</td>
      <td>Cell 4</td>
    </tr>
  </table><br>

</body>

</html>
```

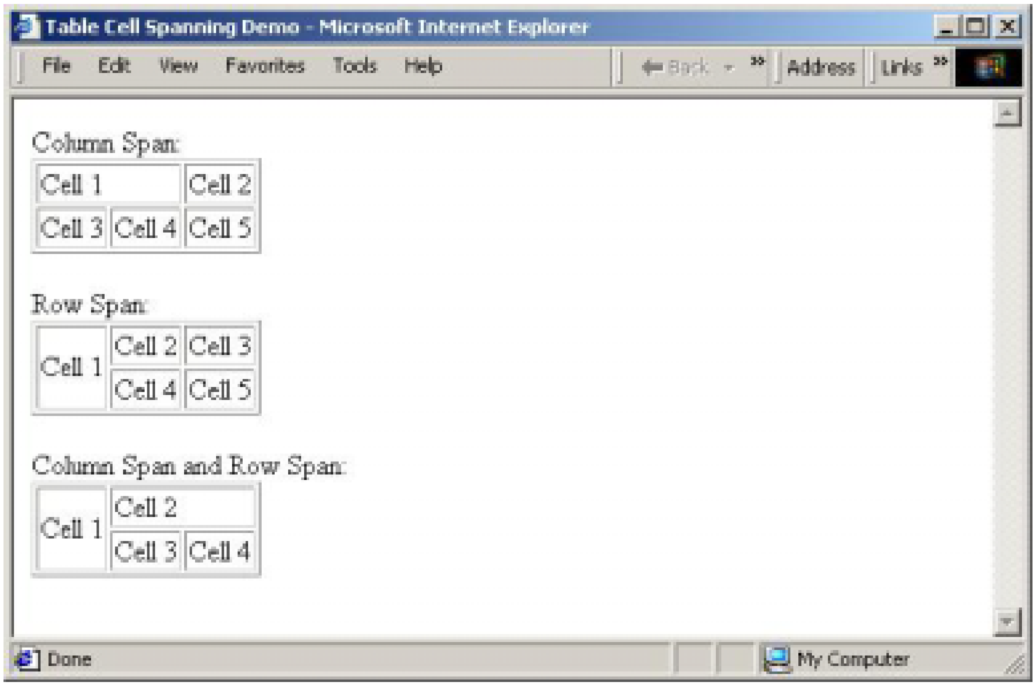


Table Cell Spanning Demo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
Links
Done
My Computer

Column Span:

<table>
<tr><td colspan="2">Cell 1</td><td>Cell 2</td></tr>
<tr><td>Cell 3</td><td>Cell 4</td><td>Cell 5</td></tr>
</table>

Row Span:

<table>
<tr><td rowspan="2">Cell 1</td><td>Cell 2</td><td>Cell 3</td></tr>
<tr><td>Cell 4</td><td>Cell 5</td></tr>
</table>

Column Span and Row Span:

<table>
<tr><td rowspan="2">Cell 1</td><td colspan="2">Cell 2</td></tr>
<tr><td>Cell 3</td><td>Cell 4</td></tr>
</table>

Insert 0015f0319.tif

Figure 3-19. Cell spanning (merging cells)

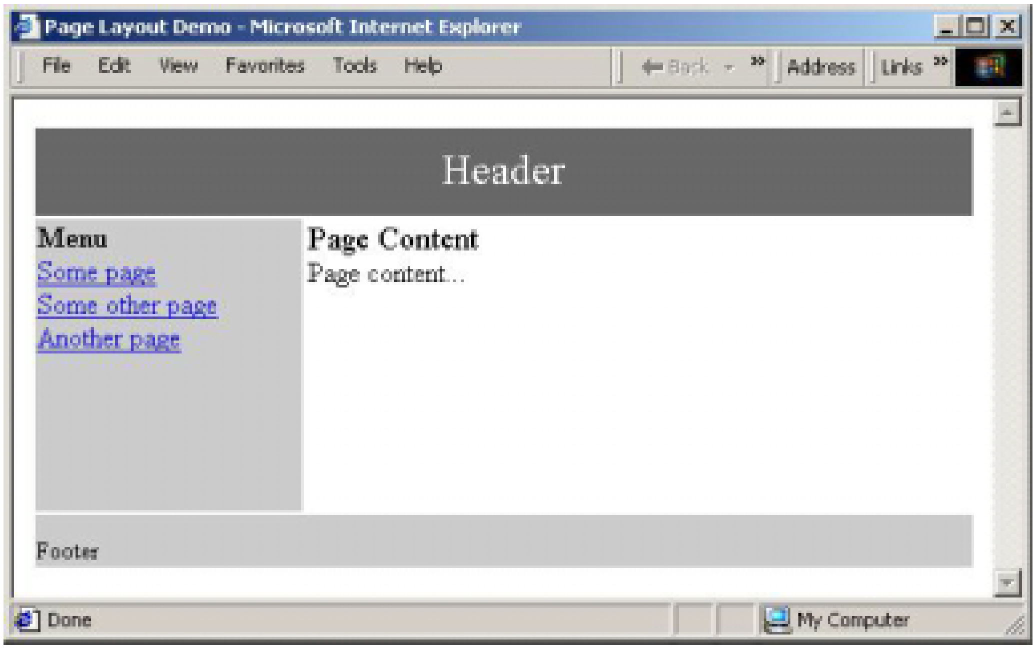
TIP Designing tables that involve multiple row and column spans can easily become very complicated. It often makes it significantly easier to plan the table on a piece of paper first before attempting to code it.

When cell spanning is used in conjunction with a borderless table and the width and height attributes, tables can be very useful tools for laying out entire pages. Listing 3-18 and Figure 3-20 show a simple example.

Listing 3-18.

```
<html>
<head>
 <title>Page Layout Demo</title>
</head>

<body>

 <table border="0" width="100%" height="100%">
  <tr>
   <td align="center" colspan="2" height="50" bgcolor="#666666">
    <font size="5" color="#FFFFFF">Header</font><br>
   </td>
  </tr>
  <tr>
```

```
   <td valign="top" width="150" bgcolor="#CCCCCC">
    <b>Menu</b><br>
    <a href="somepage.html">Some page</a><br>
    <a href="someotherpage.html">Some other page</a><br>
    <a href="anotherpage.html">Another page</a><br>
   </td>
   <td valign="top">
    <font size="4">Page Content</font><br>
    Page content...<br>
   </td>
  </tr>
  <tr>
   <td valign="bottom" colspan="2" height="30" bgcolor="#CCCCCC">
    <font size="2">Footer</font>
   </td>
  </tr>
 </table>

</body>

</html>
```



Insert 0015f0320.tif

Figure 3-20. Simple page layout using tables

When tables are used within this simple design, a significant amount of precision can be achieved in the layout. Tables, which at first glance offer nothing more than a way to display data neatly, offer a very powerful mechanism for page layout and design.

Forms

Forms were introduced in HTML to allow for a certain amount of interaction between the user and the site. There are essentially two parts to interactivity on Web sites: the user interface (which is the forms) and the actual programming. Client- and server-side scripting is where the programming comes in. This section only covers the user interface design.

The bounds of a form are defined by the form tag. Although it is possible to have multiple forms on one page in HTML, it is best to get in the habit of using only one form per page, as this is a general limitation of most ASP.NET applications. The form tag has two pertinent attributes: action and method. The action attribute defines where the data in the form must be submitted to; it is a URL value. In ASP.NET applications, this is an ASP.NET page, however, it can point to CGI applications or ISAPI applications. The method attribute describes how the data from the form must be sent to the file specified in the action attribute. It takes a value of either post or get. The difference between post and get will be dealt with later, but for now, use post unless instructed otherwise.

An HTML form is really a collection of controls where data can be inputted. HTML provides single and multiline input boxes, check boxes, radio buttons, combo boxes, and buttons. The values of these controls are submitted to the location specified in the form tag.

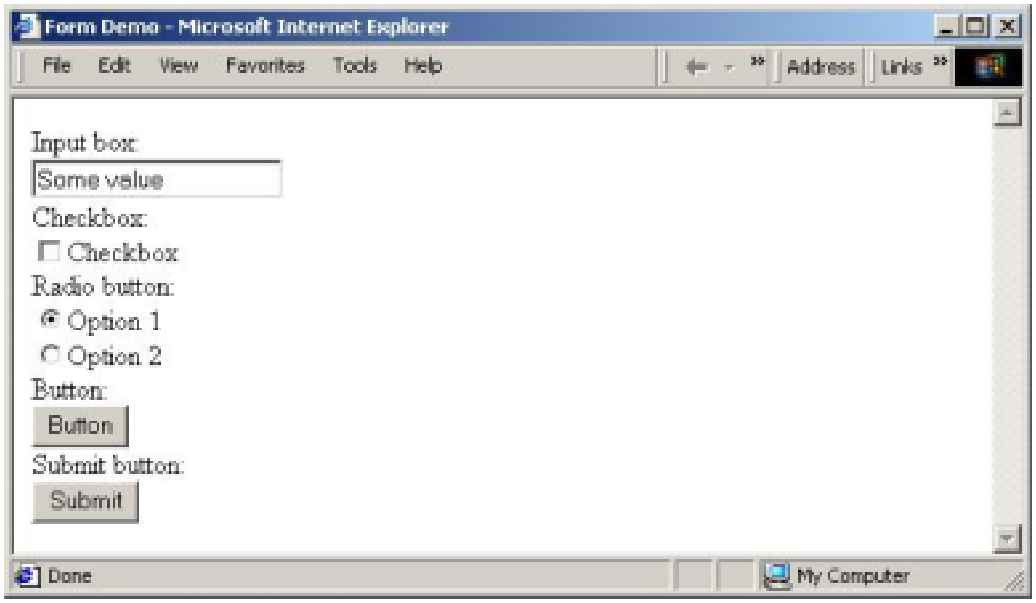
Almost all of the controls provided by HTML are inserted using the input tag. This tag has three important attributes: type, name, and value. The type attribute accepts the values text, checkbox, radio, button, and submit. The name and value properties contain the unique reference to the control and its default value, respectively. Forms normally contain at least one submit button (an input tag with the type attribute set to submit). When a user clicks the submit button, the browser sends the data in the form to the form’s action URL.

Listing 3-19 and Figure 3-21 show the use of the controls inserted using the input tag.

Listing 3-19.

```
<html>
<head>
 <title>Form Demo</title>
</head>

<body>

 <form action="http://www.yoursite.com/cgi-bin/process.pl" method="post">
 Input box:<br>
 <input type="text" name="txtInputbox" value="Some value"><br>
 Checkbox:<br>
```

```
  <input type="checkbox" name="chkCheckbox" value="on">Checkbox<br>
  Radio button:<br>
  <input type="radio" name="rdoGroup" value="option1">Option 1<br>
  <input type="radio" name="rdoGroup" value="option2">Option 2<br>
  Button:<br>
  <input type="button" name="btnButton" value="Button"><br>
  Submit button:<br>
  <input type="submit" name="btnSubmit" value="Submit"><br>
  </form>

</body>

</html>
```



Input box:

Some value

Checkbox:

☐ Checkbox

Radio button:

● Option 1

○ Option 2

Button:

Button

Submit button:

Submit

Insert 0015f0318.tif

Figure 3-21. Form controls inserted via the input tag

An interesting control is the radio button. The reason that both of the radio buttons have been assigned the same name is to define a group of options. All radio buttons with the same name will be part of that group. For example, if an online store wants to present users with an option for delivery between “Within 1 day” and “Within 1 week” and another option for payment between “Check” and “Credit card,” four radio buttons are required. The first two would have the same name, and the second pair would have a different name.

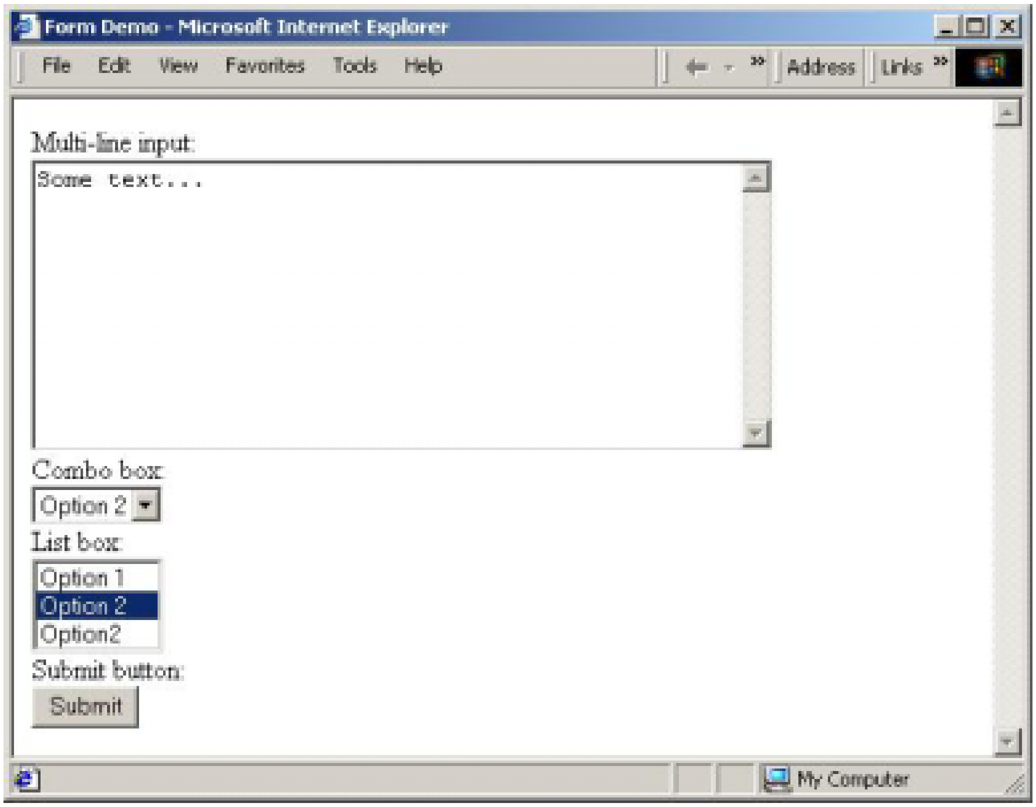
The other control that needs further explaining is the button control. It seemingly has no point—its value cannot be changed, and clicking it doesn’t render any results. This is because input buttons are typically only used in conjunction with a client-side script, which can trap the clicking action and execute code accordingly.

The two other controls are the combo box and the multiline input box (also known as the textarea). The textarea tag contains two attributes to define its size: rows and cols. It also has a name attribute. Its default value is defined by the content between its opening and closing tags.

The select tag defines a combo box. It has two important attributes: name and size. The size attribute defines the size of the combo box. In fact, it actually converts the combo box to a list box if the value specified is greater than 1. The option tag is used to insert options into the select box. These tags are inserted between the opening and closing select tags. The two important attributes are value and selected. Listing 3-20 and Figure 3-22 show the use of textarea and select.

Listing 3-20.

```
<html>
<head>
 <title>Form Demo</title>
</head>

<body>

 <form action="http://www.yousite.com/cgi-bin/process.pl" method="post">
 Multi-line input:<br>
 <textarea name="txtTextarea" rows="10" cols="50">Some
text...</textarea><br>
 Combo box:<br>
 <select name="cbo1">
  <option value="option1">Option 1</option>
  <option value="option2" selected>Option 2</option>
  <option value="option3">Option 3</option>
 </select><br>
 List box:<br>
 <select name="cbo1" size="3">
  <option value="option1">Option 1</option>
  <option value="option2" selected>Option 2</option>
  <option value="option3">Option 3</option>
 </select><br>
 Submit button:<br>
 <input type="submit" name="btnSubmit" value="Submit"><br>
 </form>

</body>

</html>
```



Some text...

Insert 0015f0322.tif

Figure 3-22. Form controls inserted via other tags

As demonstrated, the selected attribute of the option tag can only be used on one option in a select. If no option in a select has this attribute, the first option is selected.

A basic working knowledge of HTML is essential to truly understand Web applications and how they work. This section provided a very brief introduction to HTML, and though there is a lot more to HTML than what has been covered, the information here provides a solid basis to move forward on.

DHTML, Client-Side Scripting, and Style Sheets

Peter: You have several headings with little more than a line in them. Can you combine all of this content into one overview paragraph that follows under the heading above? Too many headings can make the content appear disorganized or thinly or superficially discussed. –tb

What Are They

DHTML is an acronym for *dynamic HTML*. DHTML refers to the combined use of HTML 4, client-side scripting, and style sheets.Cascading Style Sheets (CSS) is a method of applying style to HTML pages. Although it can be used in other ways, its most important function is to separate style from content and allow for much easier site administration. Client-side scripting is programming that allows for the manipulation of elements on an HTML page. All code is executed on the client side (i.e., in the browser) and can react to user actions (events) according to what the browser supports. Client-side script is totally reliant on the capabilities of the browser. Some browsers have more capabilities than others, and some have no client-side scripting capabilities at all. Examples of common tasks for client-side script are image rollovers (when the mouse cursor is moved over an image and the image changes) and form validation (checking that all required fields have been filled in before a form is submitted).

Browser Support and Incompatibilities

As I briefly discussed in the section on HTML, it is an unfortunate reality that the most commonly used Web browsers have numerous incompatibilities. Internet Explorer 4.0 and above and Netscape 4.0 and above can be said to generally support HTML 4. Pages coded using HTML 4 tags and compliant with basic HTML rules will normally look identical on both Internet Explorer 4.0 and above and Netscape 4.0 and above. The same applies to Internet Explorer 3.0 and Netscape 2.0 with regard to HTML 3.2. Basic CSS will work similarly with both Internet Explorer 4.0 and above and Netscape 4.0 and above. The problem of incompatibility really seems to strike hard when client-side script is used; however, there are also incompatibilities between Internet Explorer and Netscape's implementations of HTML and CSS, and these cannot be ignored.

Over the past few years, Web developers have devised and used several methods to try and overcome the problems presented by these incompatibilities. One very popular method that appeared shortly after the introduction of the 4.0 browsers was *browser redirection.* Using this method, two or more versions of a site were created, most commonly one for Internet Explorer and one for Netscape. The entry page for the site (the page that users first go to) worked out which browser the visitor was using and redirected him or her to the appropriate site version. This approach was fairly common among smaller sites, but it was totally unfeasible for larger sites, simply because two versions of the site had to be created. Unless content was taken from a central database, all content had to be duplicated. This redirection approach, which was in theory a good method of ensuring that a site worked in both Internet Explorer and Netscape, proved to be extremely impractical.

A method that is still fairly common today is that of *dynamic page building*. Unlike the redirection method, there is only one version of the site. However, pieces of code (particularly client-side scripting) that only work in a particular browser version are only inserted into the page if the visitor is using the correct browser. Visitors not using the particular browser receive the equivalent code guaranteed to work on most browser implementations. This method, in combination with the next one, is very effective and should be used if browser-specific code is required.

Several HTML tags have attributes that perform the same function in Netscape and Internet Explorer, but they are different in syntax, or one browser supports a nonessential attribute that the other does not support. The solution to this type of problem is made possible by one of the advantages and disadvantages of the way browsers render HTML. If Internet Explorer or Netscape find a tag or an attribute that it does not know how to display, it will simple ignore that tag or attribute. When viewed, the page will not display any error messages or the like. The reason why this is a disadvantage is that it allows people to write very, very sloppy HTML code. However, when used diligently, it can be useful to help solve one browser incompatibility problem. If the syntax of a particular attribute is different in Internet Explorer and Netscape, simply include both. If Internet Explorer is used to view the page, it will recognize the attribute that it knows and apply it, and it will simply ignore the one that it does not recognize. Netscape will behave similarly.

The lowest common denominator method is probably the best and most well-used method. Most functionality that can be achieved in Netscape via scripting will work in Internet Explorer. The two browsers' HTML and CSS implementations are very similar (more so than their respective scripting implementations). By sacrificing and not using proprietary functionality in either browser, which ultimately does not amount to a significant proportion, Web sites can be made to be almost completely cross-browser compatible without the pain of extra programming or the extreme of developing different versions of sites for different browsers.

Whichever method is used to ensure cross-browser compatibility, all Web applications must be thoroughly tested in a wide variety of environments to make sure that everything is rendered and behaves as it is designed to. There is no substitute for comprehensive testing. All applications must go through a phase of intensive testing before deployment.

The subject of developing cross-browser compatible applications is very large, and this brief section highlighted only some of the methods available. These methods, incidentally, are used internally by ASP.NET in certain cases, and the book *Writing Cross-Browser DHTML* by Heather Williamson (Apress, 2001) presents a much more comprehensive look at the topic.

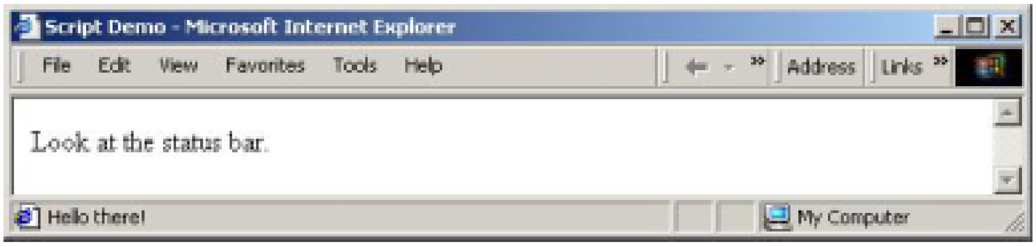
Client-Side JavaScript Primer

Netscape and Internet Explorer both support client-side scripting, but one of the most important differences between the two is that Internet Explorer supports both VBScript (Visual Basic Script, a subset of the Visual Basic language) and JScript (Microsoft's implementation of JavaScript) as scripting languages that can be used, whereas Netscape only supports JavaScript. For any site that needs to run on Netscape as well as Internet Explorer, JavaScript is naturally the only choice. Another important fact to note is that not all versions of Internet Explorer and Netscape support client-side scripting (i.e., versions 2.0 and below) and, more pertinent to the issue, the difference between the capabilities of client-side scripting in the 3.0 browsers and the 4.0 browsers is very large. Scripting capabilities in the 3.0 browsers is very elementary, and it is for this reason that most scripting authors simply ignore the 3.0 browsers and instruct script not to run in 3.0 browsers. The next important point is that unlike HTML rendition in browsers where errors are simply ignored, client-side scripting errors produce runtime errors that are displayed in a message box in the browser when the code is executed.

Client-side scripting code is embedded into HTML pages using the script tag. The script tag has a language attribute for defining the scripting language to be used; however, JavaScript is the default on both Netscape and Internet Explorer, so this does not need to be explicitly set. JavaScript's syntax is very closely based upon that of Java, although there are several important differences. These differences will become apparent to Java programmers as they work through the few examples in this section. Owing to the nature of Java syntax, JavaScript will also look very familiar to C/C++ programmers. Listing 3-21 and Figure 3-23 show a simple example of JavaScript.

Listing 3-21.

```
<html>
<head>
 <title>Script Demo</title>
 <script>
 <!--
  window.status = "Hello there!";
 -->
 </script>
</head>

<body>

 Look at the status bar.

</body>

</html>
```



Look at the status bar.

Hello there!

Insert 0015f0323.tif

Figure 3-23. Client-side script using JavaScript to change the browser’s status bar

text

The first unusual thing about this example is the use of the “tag” <!-- --> inside the script block. This is actually an HTML comment. Anything inside it will be ignored by the HTML processor. The reason the comment is included is so if the page is viewed using a browser that does not support client-side script, the code will not be displayed, as the HTML processor will take the code as a comment.

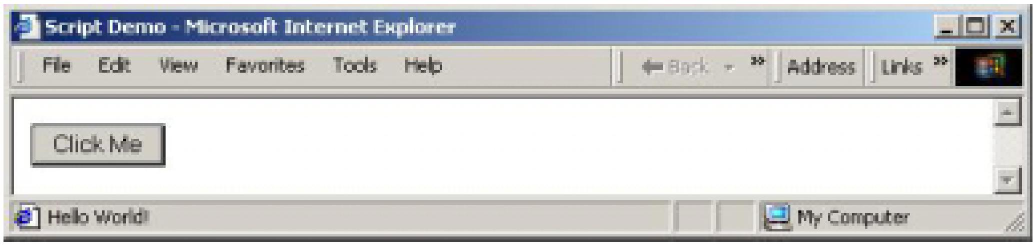
Probably the next most interesting part is that there is no main() function, no event handlers, and no other structures. Client-side script was intended to be quick and easy, and as such, it is not very involved; there is no requirement for classes or the like. There are also no library “imports.” Everything that can be done using client-side script is automatically made available.

Internet Explorer and Netscape allow client-side script to have access to and manipulate many elements of an HTML page, as well as some elements of the browser itself. In Listing 3-21, the window object is used. You can think of the window object as the object that represents the browser and the functionality exposed by it, as well as the container for the HTML document. As you can see, the status property refers to the text of the status bar in the browser. The assignment operator is the equal sign (=) in JavaScript, and statements must end with a semicolon (;). The delimiter for strings is the quotation mark (").

Many HTML elements have event attributes where JavaScript code can be inserted. JavaScript also has support for user functions, which can return values or not. Listing 3-22 and Figure 3-24 demonstrate HTML events and a simple function working together.

Listing 3-22.

```
<html>
<head>
 <title>Script Demo</title>
 <script>
 <!--
  function changeWindowStatus(newstatus) {
   window.status = newstatus;
  }
 -->
 </script>
</head>

<body>

 <form>
  <input type="button" name="Button1" value="Click Me"
onClick="changeWindowStatus('Hello World!');">
 </form>

</body>

</html>
```



Hello World!

Insert 0015f0324.tif

Figure 3-24. HTML events and JavaScript user functions in action

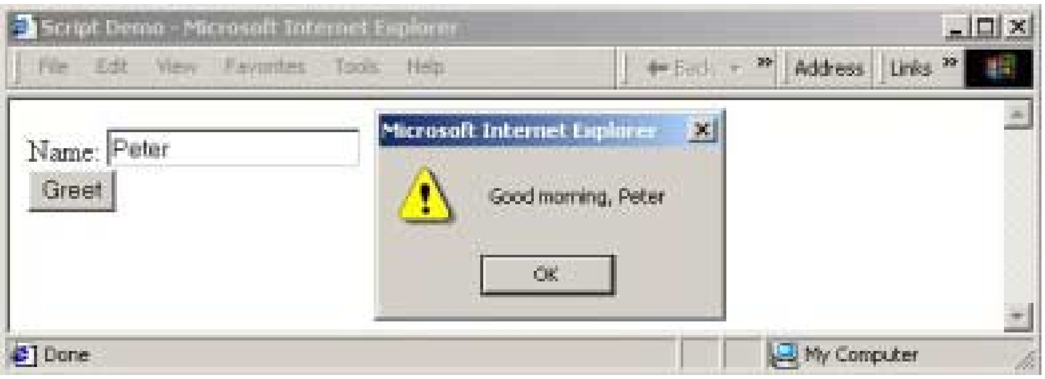
When the button is clicked, the function changeWindowStatus is called, which in turn changes the browser’s status bar. A couple of interesting things you can learn from this example are that JavaScript is a typeless language, and when JavaScript is called via an HTML event, the string delimiter is the apostrophe ('). Because JavaScript is typeless, the definition of functions is slightly different from Java and C/C++. Additionally, parameter definitions in functions do not have specific types assigned to them. Programmers who are not familiar with C-style syntax will not recognize the curly brace characters ({}). In JavaScript, curly braces are block delimiters.

Listing 3-23 and Figure 3-25 show several other features of JavaScript, including conditions, variable declaration, and its capability to access HTML elements.

Listing 3-23.

```
<html>
<head>
 <title>Script Demo</title>
 <script>
 <!--
  function SayHello() {
   var name;
   var today;
   name = document.form1.txtName.value;
   today = new Date();
   if (today.getHours() < 12) {
    window.alert("Good morning, " + name);
   } else {
    window.alert("Hello, " + name);
   }
  }
 -->
 </script>
</head>

<body>

 <form name="form1">
  Name: <input type="text" name="txtName"><br>
  <input type="button" value="Greet" onClick="SayHello();">
 </form>

</body>
```

```
</html>
```


Script Demo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
Links
Name: Peter
Greet
Microsoft Internet Explorer
OK
Done
My Computer

Good morning, Peter

Insert 0015f0325.tif

Figure 3-25. Working with dates, conditions, and HTML elements

Variables are declared using the var keyword. Again, this is different from Java and C/C++ because JavaScript is typeless. The value of the input box is accessed via the document object, which is the container of the HTML document. In the same line, form1 refers to the name of the form that the field is in, and finally txtName is the name of the object to access. The value is a property of most form elements, which contains the element's current value.

The new keyword is used for instantiation of internal JavaScript objects—in this case, the Date object. When created, the Date object's default value is the current date and time. The if condition requires that parentheses are placed around the condition to be tested. The Date object today contains the method getHours, which returns the hour. The comparison operator used in this case is "less than or equal to" (the others available are "equal to," which is ==; "not equal to," represented by !=; and "greater than or equal to," represented by >=). Again, the curly braces define code blocks in JavaScript. The alert method of the window object displays a message box. String concatenation in JavaScript is performed via the + operator.

There is a lot more to JavaScript than this section has covered. However, this section should suffice as a primer in understanding a small bit of what client-side scripting can do. Sooner or later, you will get a project where client-side script is the only sensible solution to some problems or challenges of it, so having a basic understanding of what it is, what it can do, and how to use it is essential.

CSS

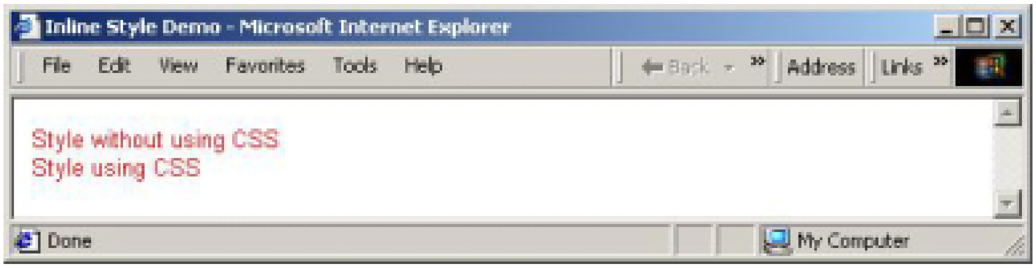
Style sheets are actually a very simple concept. Their primary purpose is to remove all style information from content in HTML documents. The advantages of doing so are far from apparent in a five-page site, but when they are used in any sizeable project, the time saved by style sheets can actually be astounding. In addition to providing productivity gains, style sheets also provide unprecedented control over the rendering of most HTML elements—control that is simply not available via normal HTML tags and attributes.

There are essentially two ways in which CSS can be applied to HTML elements. The first is via inline style, and the second is via a style sheet, which can either be embedded in the document or in an external file.

All HTML elements that can be manipulated via CSS have a style attribute. The value of this attribute is all the CSS statements that need to be applied to that particular element. CSS statements closely resemble, yet are not always identical to, their HTML counterparts with regard to naming. CSS statements are totally different from HTML in syntax, however, as they do not use tags, but rather “property: value” pairs. Listing 3-24 and Figure 3-26 show a basic example of how CSS can replace HTML styling.

Listing 3-24.

```
<html>
<head>
 <title>Inline Style Demo</title>
</head>

<body>

 <div>
  <font face="Arial" size="2" color="#FF0000">
   Style without using CSS
  </font>
 </div>

 <div style="font-family:Arial;font-size:10pt;color:#FF0000;">
  Style using CSS
 </div>

</body>

</html>
```



Style without using CSS
Style using CSS

Insert 0015f0326.tif

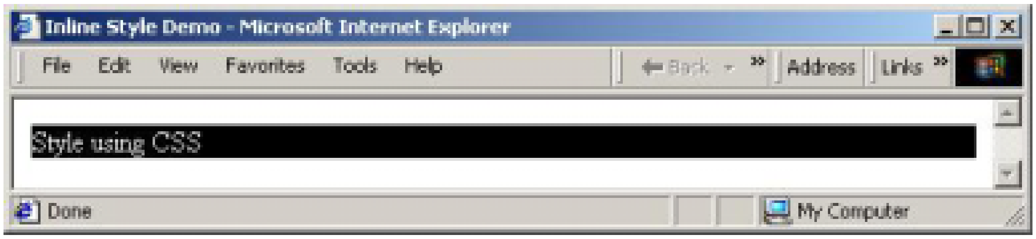
Figure 3-26. Applying style using inline styles

The “property: value” pairs in the style attribute are separated by semicolons. As you can see, properties that only apply to certain possible parts of an HTML element’s contents, such as the font, are separated from properties that can apply to all the content, such as color, which can apply to the text, borders, and so forth. The next important thing to notice is that CSS properties occasionally take values in different units from those in HTML. In this example, the font-size property takes values in points (pt, the standard measurement for fonts), rather than the nonstandard units of HTML.

This example neither separates style from content nor provides any extra control over the HTML element that is not available via HTML. Listing 3-25 and Figure 3-27 show how you can set a division’s background color using inline style, which is not possible using HTML tags or attributes.

Listing 3-25.

```
<html>
<head>
 <title>Inline Style Demo</title>
</head>

<body>

 <div style="background-color:#000000;color:#FFFFFF;">
  Style using CSS
 </div>

</body>

</html>
```



Insert 0015f0327.tif

Figure 3-27. Applying a background color using inline style

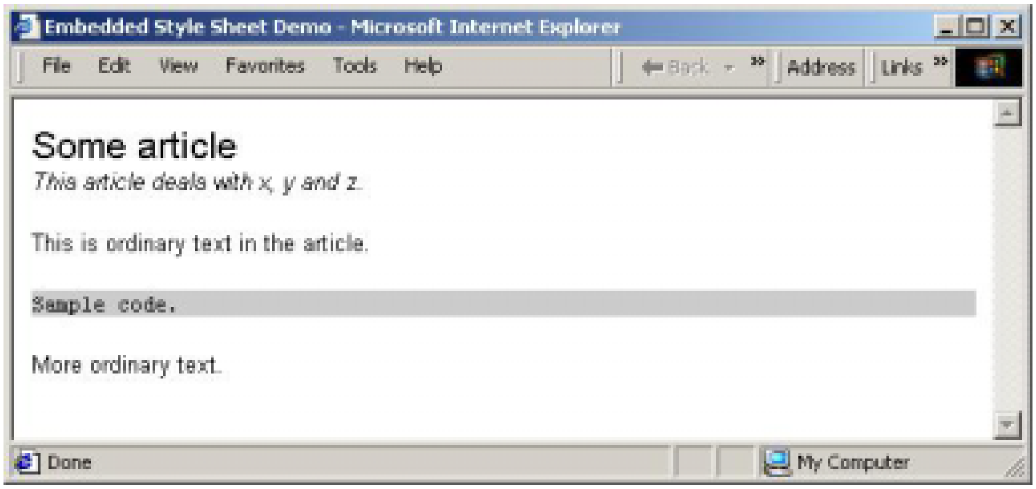
An area in which inline style is very appropriate is the formatting of a particular HTML element, which is not possible via normal HTML attributes, and there are only a few isolated cases where this formatting needs to be done. However, if formatting needs to be done regularly, creating a style sheet is a more appropriate and useful solution.

You can use style sheets in several ways. You can define styles where all instances of a particular HTML tag (e.g., div) have that style applied to them. You can also define styles that are only applied to tags that specifically “ask” for them to be applied. As I mentioned earlier, style sheets come in two flavors: embedded and external. Listing 3-26 and Figure 3-28 demonstrate the use of an embedded style sheet with different types of style definitions.

Listing 3-26.

```
<html>
<head>
  <title>Embedded Style Sheet Demo</title>
  <style>
  <!--
    div {
      font-family: Arial;
      font-size: 10pt;
    }

    .code {
      font-family: Courier New;
      font-size: 9pt;
      background-color: #CCCCCC;
    }

    .header {
      font-size: 16pt;
    }

    #abstract {
      font-style: italic;
    }
-->
</style>
</head>

<body>

  <div class="header">
    Some article
  </div>
  <div id="abstract">
    This article deals with x, y and z.
  </div><br>
  <div>
    This is ordinary text in the article.
  </div><br>
  <div class="code">
```

```
     Sample code.
   </div><br>
   <div>
     More ordinary text.
   </div>

 </body>

 </html>
```



Embedded Style Sheet Demo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
Links
Done
My Computer

Some article

This article deals with x, y and z.

This is ordinary text in the article.

`Sample code.`

More ordinary text.

Insert 0015f0328.tif

Figure 3-28. Using embedded style sheets to define formatting and style

As you can see, style sheets are embedded in style tags. As with script, an HTML comment is wrapped around all the CSS code, so that should the browser not support CSS, the code will not be displayed. The first block of code is a typical example of CSS:

```
 div {
  font-family: Arial;
  font-size: 10pt;
 }
```

This code instructs all division elements to be rendered with the text formatted in 10-point Arial font, unless those settings are overridden. Therefore, all text within div tags use 10-point Arial font unless otherwise instructed (by font tags or other, more specific styles). The div in this code block could be replaced with any HTML element that accepts CSS, such as body, span, or even td.

The HTML 4 specification makes allowance for a class attribute, which every element that is accessible via CSS has. This class attribute allows distinction between a particular element and other elements, so that in this example only some division elements are formatted in a certain way:

```
 .header {
  font-size: 16pt;
```

```
}
```

The period preceding “header” indicates that the style rules inside this CSS block must only be applied to elements that have a class="header" attribute. In the example, this was applied to a division, and because it applies more specifically, the original font-size value of a div is ignored. The same applies to the .code selector (a selector is the name given to the part of the CSS block that specifies what it applies to). The hash mark (#) in the #abstract selector means that the style rules in that block will be applied to all tags with an attribute of id="abstract ". This example has only been included to demonstrate what the hash mark means in a selector; however, the class method is preferable, as the id attribute of tags is often analogous to the name attribute, which may cause problems in both client- and server-side scripts.

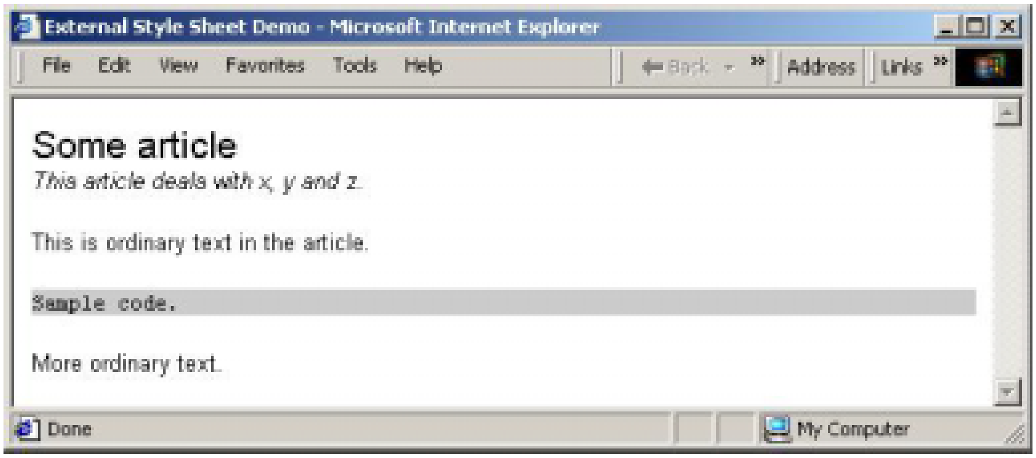
The primary advantage of using a style sheet is that if one property in a block is changed, the changes are immediately effected in the entire document where those rules are used. However, going back to Listing 3-26, if there were lots of articles on the site all using that format (and thus style sheet), it would still be a major time-consuming effort if, say, the background color of the code regions needed to be made #999999 instead of #CCCCCC. Each individual article would have to be opened and the style sheet modified accordingly. This seems rather silly, as all the documents use the same style sheet. The solution is an external style sheet. An external style sheet is a plain text file with the CSS code normally inside a style element residing in it. Each document that uses this style sheet creates a reference to it in the HTML code so that it is included. HTML provides the link tag for this purpose. The major advantage of using an external style sheet is that it only takes one change in one file for all the HTML documents using it to be updated. Listings 3-27a and 3-27b (CSS and HTML files respectively) and Figure 3-29 demonstrate how an external style sheet would be used for Listing 3-26.

Listing 3.27a

```
div {
 font-family: Arial;
 font-size: 10pt;
}

.code {
 font-family: Courier New;
 font-size: 9pt;
 background-color: #CCCCCC;
}

.header {
 font-size: 16pt;
}

#abstract {
 font-style: italic;
}
```

Listing 3.27b

```
<html>
<head>
 <title>External Style Sheet Demo</title>
 <link rel="stylesheet" href="style.css">
</head>

<body>

 <div class="header">
  Some article
 </div>
 <div id="abstract">
  This article deals with x, y and z.
 </div><br>
 <div>
  This is ordinary text in the article.
 </div><br>
 <div class="code">
  Sample code.
 </div><br>
 <div>
  More ordinary text.
 </div>

</body>

</html>
```



Some article

This article deals with x, y and z.

This is ordinary text in the article.

```
Sample code.
```

More ordinary text.

Insert 0015f0329.tif

Figure 3-29. Using external style sheets

As you can see, adding in an external style sheet to a document is a trivial process. Building a standard style sheet for a site and then including it in each page is normally a simple way of obtaining a consistent look for pages in the site.

Again, this section provided only a very small look at CSS, what it is, what it can do, and how it's done. CSS is commonly used and a solid understanding of it is essential. Visual Studio .NET includes an external style sheet in Web Forms by default, and this section should suffice in explaining all of what is included in the base style sheet.

Web Servers

The term "Web server" can be used in the context of a computer and a piece of software. A *Web server,* for the purposes of this book, is a computer that is permanently connected to the Internet and runs software such that when a request for a file is received, it returns the requested file. The machine itself does not have to be very different from powerful desktop machines. Entry-level servers normally have CPUs, memory, and hard drives similar to those of high-end workstations. Servers do not normally feature advanced multimedia functionality and will generally have bottom-of-the-range video adapters and no sound cards. Microsoft's customers generally opt for the use of a "scale out", rather than "scale up" policy to achieve scalability (the capability of a Web server to serve many simultaneous users) In a *scale up* policy, more users are catered to by adding more CPUs and memory to the server. In a *scale out* policy, if an application needs to serve more users, a server is simply added to the farm as part of a load-balancing environment.

> **NOTE** A *server farm* is a collection of servers, normally in the same physical location, that are linked together. *Load balancing* is functionality in a server farm where, when a request is received, the server with the least load is found and the request is sent to that machine.

Using this scale out policy, farms serving applications using Microsoft technologies rarely have servers with more than two processors and 1GB of RAM. When the current servers in a farm are not adequate to efficiently serve the requests, more servers are added to the farm.

An extremely important factor in serving Web applications is Internet connectivity bandwidth. Server farms should not be connected to the Internet via dial-ups, ISDN, DSL, or cable connections. They should be connected via fast, permanent, reliable connections—at a minimum, a single T1 or T3 line.

In a Microsoft environment, the Web server software used is Internet Information Server (IIS), which ships with Windows 2000 Server and above. This is the software that intercepts requests on the HTTP (Web) TCP/IP port 80. It then interprets the request and sends a response back to the user that made the request with the appropriate data. The full version of IIS, which ships with Windows 2000 Server, has the capability to serve up multiple sites on different domains. However, Windows 2000 Professional ships with a version of IIS that has the limitation of only being able to serve one site on one domain. This version is perfectly adequate for the purposes of ASP.NET development, which obviously requires IIS, because it is essentially an “add-on” to a Web server. (This is why Chapter 2 provided instructions for installing IIS.)

An important fact to know is that although Web browsers only render HTML documents, which are the format of Web documents, IIS and other Web server software can serve up any documents, not only HTML documents—for example, zip archives. How the user’s browser handles this depends on the browser, but Netscape and Internet Explorer both handle responses from the Web server similarly. If the response is HTML, it is rendered in the browser. If the response is not HTML, a dialog box displays that asks where the user would like to save the incoming file to on his or her hard drive.

> **NOTE** sThe default browser behavior of opening up a ‘Save As’ dialog when a non-HTML file is opened is not the case in all instances. For example, Internet Explorer will display Microsoft Office documents in a special mode of the browser if the server sends an Office document. However, for the most part, non-HTML files result in a Save As dialog box being displayed.

Server-Side Programming

Chapter 1 covered the concept of server-side programming. To recap, in server-side programming, all code is executed on the server and the result is sent to the client. In this book, server-side programming consists of requests to the server, which take the form of HTTP requests; the code executed on the server, which is ASP.NET code using the Visual Basic .NET language; and the result, which is HTML, or DHTML if the client supports it, which is again transported to the client using a standard HTTP response. From the client’s viewpoint, it is sending a request for a file and receiving a response—no different from any other HTML file that it requests. The execution of the code on the server is completely transparent.

This chapter briefly introduced client-side scripting in HTML documents, in which JavaScript code is sent with the HTML and is executed by the browser when it is received. Here is a brief summary of the advantages and disadvantages of server-side scripting as compared to client-side scripting.

Advantages:

* No code (other than the resulting [D]HTML) is sent to the browser, and thus all code is secure—there is no way that people can view it.
* Server-side scripts are able to make use of components and .NET infrastructure on the server, without the .NET Framework having to be installed on the client machine.
* Server-side scripts are able to access resources on the server (farm). Such resources could include SQL Server databases, XML, and text files. Only the parts of these resources that the script specifies must be sent.
* Because the result that the Web server returns to the client is plain HTML, server-side applications can be guaranteed to work on all browsers, unlike client-side applications, where the script will only be executed if the browser supports client-side scripting.

- Pages can be built dynamically according to the browser the client is using to request a particular script, and thus DHTML will be returned if the client is using a browser that supports DHTML, and plain HTML will be returned if the client is using a browser that does not support DHTML.

Disadvantages:

- All interaction between the user and the application requires the user to submit information, which is then sent to the server, processed, and returned. This makes interaction slow, due to the fact that the Internet is relatively slow. Client-side scripts can execute immediately.
- Processing scripts on the server use up server resources and thus make server upgrades and additions more prevalent than in client-side–only applications.

Where ASP.NET Fits In

ASP.NET is essentially an “add-on” to Microsoft IIS and provides server-side programming capabilities to IIS. ASP.NET and the rest of the .NET Framework takes almost all of the work of converting code to HTML away from the developer, leaving him or her to only worry about the functionality and back-end logic of the application. The “plumbing” is done by ASP.NET. As ASP.NET applications are just another type of .NET application, any language can be used, although this book uses the Visual Basic .NET language.

Where VB .NET Fits In

To clarify, when this book refers to Visual Basic .NET (VB .NET), unless otherwise explicitly stated, it is referring to the VB .NET language and not the development environment. Because VB .NET applications (Web Forms, Web Services, WinForms, and so on) are developed in the Visual Studio .NET IDE, this book uses the term Visual Studio .NET (VS .NET) to refer to the development environment. Thus, this section deals with VB .NET, the language.

VB .NET is actually shipped with the .NET Framework, and VS .NET is not required to use it. However, VS .NET provides an extremely productive environment for VB .NET development, and among many other things, it makes the transition from traditional desktop development to Web application development much simpler for Visual Basic developers.

ASP.NET is not a language. You can think of it as an interface between complicated Web server internals and a programming language, with a multitude of functionality built in. In this book, the programming language used is VB .NET, although you could use any other .NET language. Whatever language is used, its purpose is essentially to access and manipulate the functionality provided by ASP.NET, relevant classes (you can think of classes as analogous to objects for now) in the .NET Framework’s base class library (BCL), and third-party classes.

VB .NET Primer

This primer is intended for non–Visual Basic developers, or VBScript developers who have never used the Visual Basic language before. Visual Basic developers may find some new items, such as initializers and structured error handling.

Variable Declaration and Data Types

VBScript is a *typeless* scripting language, meaning that it does not explicitly differentiate between Integers, Strings or any other data types, and all necessary type conversions are performed automatically. VB .NET is completely on the other end of the scale. It is very strongly typed—all variables must be declared with a type (no variant variables types), and all conversions must be made explicitly, otherwise a runtime error is raised (this behavior can be limited to a certain extent to help backward compatibility with original Visual Basic code). Microsoft introduced strong types to help reduce the amount of bugs caused by type problems.

Variable declaration in VB .NET is performed via the Dim keyword. The basic syntax is as follows:

```
Dim VarName As Type
```

Table 3-1 shows the data types supported by VB .NET.

Table 3-1. Important Visual Basic .NET Data Types

VB .NET Type	**Storage Size**	**Value Range**
Boolean	4 bytes	True or False
Byte	1 byte	0–255 (unsigned)
Char	2 bytes	0–65535 (unsigned)
Date	8 bytes	Jan 1, 1 CE to Dec 31, 9999
Decimal	12 bytes	+/–79,228,162,514,264,337,593,543,950,335 with no decimal point; +/–7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is +/–0.0000000000000000000000000001
Double (double-precision floating-point)	8 bytes	–1.79769313486231E308 to –4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Integer	4 bytes	–2,147,483,648 to 2,147,483,647
Long (long integer)	8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Short	2 bytes	–32,768 to 32,767
Single	4 bytes	–3.402823E38 to –1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
String	10 + (2×length of string) bytes	0 to approximately 2 billion Unicode characters

VB .NET also supports initializers, so variables can have a value set to them in the same line as their declaration. The syntax for initializers is as follows:

```
Dim VarName As Type = Value
```

VB .NET also has support for arrays, which are declared using the Dim statement. The size of arrays cannot be fixed and the ReDim statement can be used to modify array size. You can declare arrays using either of the following:

```
Dim ArrName(Size) As Type
Dim ArrName() As Type = {InitialValue1, InitialValue2..., InitialValueN}
```

Although array size is not constant, the number of dimensions must be fixed. The preceding examples create one-dimensional arrays. Multidimensional arrays are created using the following syntax:

```
Dim ArrName(Dimension1Size, Dimension2Size..., DimensionNSize) As
Type
```

Operators

VB .NET contains several assignment operators, many similar to those found in C/C++-style languages. Table 3-2 shows all the assignment operators available, their syntax, and their purpose.

Table 3-2. Visual Basic .NET Assignment Operators

Operator	**Syntax**	**Purpose**
=	Variable = value	Variable/property assignment
^=	Variable ^= expression	Raises the value of the variable to the power of “expression” and assigns the result to the variable
*=	Variable *= expression	Multiplies the value of the variable by “expression” and assigns the result to the variable
/=	Variable /= expression	Divides the value of the variable by “expression” and assigns the result to the variable
\=	Variable \= expression	Divides the value of the variable by “expression” and assigns the integer part of the result to the variable
+=	Variable += expression	Adds “expression” to the value of the variable and assigns the result to the variable
-=	Variable -= expression	Subtracts “expression” from the value of the variable and assigns the result to the variable
&=	Variable &= expression	Concatenates “expression” with the value of the variable and assigns the result to the variable

Table 3-3 shows the comparison operators for use in If statements and the like.

Table 3-3. Visual Basic .NET Comparison Operators

Operator	**True If**	**False If**
< (less than)	expression1 < expression2	expression1 >= expression2
<= (less than or equal to)	expression1 <= expression2	expression1 > expression2
> (greater than)	expression1 > expression2	expression1 <= expression2
>= (greater than or equal to)	expression1 >= expression2	expression1 < expression2
= (equal to)	expression1 = expression2	expression1 <> expression2
<> (not equal to)	expression1 <> expression2	expression1 = expression2

A full complement of arithmetic operators is also included, as shown in Table 3-4.

Table 3-4. Visual Basic .NET Arithmetic Operators

Operator	**Syntax**	**Purpose**
^	Result = number^exponent	Raises a number to the power of an exponent
*	Result = number1*number2	Multiplication
/	Result = number1/number2	Division
\	Result = number1\number2	Division (returns Integer)
Mod	Result = number1 Mod number2	Modulo (remainder)
+	Result = expression1+expression2	Addition or concatenation
-	Result = number1-number2	Subtraction

Basic Control Structures

VB .NET provides the If statement as a basic level of flow control. The basic structure of If statements is as follows:

```
If condition Then
 Statements
End If
```

However, the If statement provides both the ElseIf and Else options to extend the basic If statement, as follows:

```
If condition Then
 Statements
Else
 Statements
End If
```

or

```
If condition Then
 Statements
ElseIf condition Then
 Statements
ElseIf condition Then
 Statements
End If
```

or

```
If condition Then
 Statements
ElseIf condition Then
 Statements
Else
 Statements
End If
```

VB .NET provides a multiple If structure similar to the switch structure in C/C++. The basic structure of a Select statement is as follows:

```
Select Case expression
  Case condition1..., conditionN
    Statements
  Case Else
    Statements
End Select
```

Unlike the C/C++ switch structure, the Select structure does not require an Else section.

There are three primary looping structures for VB .NET, each with its own slight variation. The For loop is an iteration loop and can take one of two forms:

```
For counter = start To end
  Statements
Next counter
```

or

```
For Each item In collection
  Statements
Next
```

The Do loop is a conditional loop that has several variations:

```
Do While condition
  Statements
Loop
```

or

```
Do Until condition
  Statements
Loop
```

or

```
Do
  Statements
Loop While condition
```

or

```
Do
  Statements
Loop Until condition
```

The last looping structure, the While loop, is also a conditional loop:

```
While condition
  Statements
End While
```

VBScript and VB programmers should note that the Wend keyword has been replaced by End While, although VS .NET makes this change automatically.

Subroutines and Functions

There are different structures in VB .NET to define procedures that return values and those that don't. Functions that don't return values (void functions in C/C++) are created using the Sub structure:

```
Sub SubName(param1 As Type..., paramN As Type)
 Statements
End Sub
```

Functions that return values use the Function structure:

```
Function FunctionName(param1 As Type..., paramN As Type) As Type
 Statements
 Return ReturnValue
End Function
```

The Return statement is used to define the result of the function. This can be replaced with a statement assigning the result to the name of the function. The type of the return is defined by the As Type after the argument list parentheses.

When functions or subs are called from code, parentheses must be included unless no arguments are taken, in which case the parentheses can be omitted. This is a change from VBScript and VB 6.0, where parentheses were not required on calls to subs, even if they had arguments. Subs and functions are both called simply by using the name thereof, followed directly by a pair of parentheses in which parameters are placed, separated by commas. The following are examples of valid sub and function calls:

```
A = SomeFunction(x, y, z)
B = AnotherFunction
C = AnotherFunction()
SomeSub(x, y)
AnotherSub()
AnotherSub
```

This codeassumes that AnotherFunction and AnotherSub do not take any parameters.

By default, parameters are passed by value. However, the ByVal and ByRef keywords are available for sub and function parameter definitions to define whether particular parameters should be passed by value or by reference. The follow example shows the correct syntax:

```
Function FunctionName(ByVal param1 As Type, ByRef param2 As Type) As
Type
 Return ReturnValue
End Function
```

NOTE The difference between by value and by reference parameter passing is subtle, but important. Where parameters are passed by reference, the actual variable being passed is sent—the function or sub has direct access to that variable and can access it as well as modify it. Parameters passed by value only receive a copy of the value, not a reference to a variable, and thus cannot modify the variable.

Comments

Comments are denoted by an apostrophe (') and a Rem statement. A Rem statement can only be used on its own line, whereas the apostrophe can be used at the end of a line of code. The syntax for both is as follows:

```
Rem This is a comment
Statement ' Comment about statement
' Comment on its own line
```

VB .NET does not have a multiline comment.

Member Scope

Member scope is important in the development of classes. It defines which variables and properties of the class can be accessed, and from where. The actual development and role of classes in VB .NET and ASP.NET is discussed in more detail later. Table 3-5 outlines the keywords in VB .NET that relate to defining member scope.

Table 3-5. Member Scope Keywords

Keyword	**Syntax**	**Purpose**
Public	Public VarName As Type	Accessible from any project where the class is referenced
Protected	Protected VarName As Type	Accessible only in the class and derived classes
Friend	Friend VarName As Type	Accessible from anywhere in the same assembly
Private	Private VarName As Type	Accessible only from within the same class

Error Handling

Visual Basic previously provided an On Error statement in conjunction with labels to provide error handling. VB .NET introduces a completely new error-handling structure that is very similar to that provided in Java and C++. The following code is a basic representation of the syntax:

```
Try
  'Statements that may cause exceptions
Catch exception1 As Type
  'Statements to deal with exception
Catch exceptionN As Type
  'Statements to deal with exception
Finally
  'Statements to perform cleanup
End Try
```

This is a very powerful structure that enables different exceptions to be handled separately. Multiple Try structures can be embedded to provide extensive error handling for all purposes. Current Visual Basic programmers might be tempted not to bother learning how to use this structured error handling, but the advantages of it over the previous inelegant, unstructured approach are so great that in the best interests of creating maintainable code that is easy to debug, you should never again have to resort to the previous On Error statements.

OOP and VB .NET Concepts Primer

OOP is an acronym for *object-oriented programming*. Simply put, it involves several programming concepts that relate to classes and objects, especially with regard to code reuse. Some of OOP's most important concepts are the basis of the .NET Framework and the base class library (BCL); thus, understanding OOP concepts goes a long way toward understanding many of the changes in VB .NET and the way .NET programming works in general.

Method Overloading

Method overloading is not an OOP concept—it is a new capability that VB .NET has and the .NET Framework uses commonly. Method overloading is used when a procedure (function or sub) could take arguments with different types. Without method overloading, this eventuality would result in multiple methods being made, with their names reflecting the arguments that they take. Listing 3-28 shows a simple example of this.

Listing 3-28.

```
' Procedures in some class
Sub DisplayInt(ByVal value As Integer)
 ' Code
End Sub
Sub DisplayString(ByVal value As String)
 ' Code
End Sub

' Code calling procedures
Dim a As Integer
Dim b As String

a = 5
DisplayInt(a)
b = "Hello"
DisplayString(b)
```

Listing 3-28 could be modified to use method overloading, as shown in Listing 3-29. This makes it easier to call the methods, as a method that performs one operation has one name, and the compiler decides which version of it needs to be called according to the parameters passed to it.

Listing 3-29.

```
' Procedures in some class
Overloads Sub Display(ByVal value As Integer)
```

```
  ' Code
 End Sub
 Overloads Sub Display(ByVal value As String)
  ' Code
 End Sub

' Code calling procedures
 Dim a As Integer
 Dim b As String

 a = 5
 Display(a)
 b = "Hello"
 Display(b)
```

Classes

A *class* is a data structure with related code to perform the functionality of the class. Classes are normally built to perform specific operations and are designed to be reusable. Classes can contain methods, properties, and variables whose scopes are defined using the keywords detailed in the previous section. Another feature of classes in VB .NET is that they can contain constructors that take parameters.

A common use for classes in ASP.NET is to to hold all shared business logic. Functionality such as authentication, which is required throughout the application, may be included in this class. A class could also be used to perform the data access for an application—this class would work with the business logic class.

The terms “class” and “object” are normally used interchangeably, but there is a difference between the two. An object is a class that has been instantiated, so technically a File “object” is not an object unless an instance of it has been created. It is simply a class—a piece of code defining a data structure with associated functionality.

Inheritance

One of the core principals of OOP is inheritance. *Inheritance* is the capability of a class to be derived from another class. In other words, a new class can start off by “inheriting” all the properties and methods of another class and then adding its own. This has several advantages. Namely, if a few classes are needed that all perform the same set of functions, as well as a few of their own, a base class can be created with these common properties and methods, and then the other classes can all inherit from the base class and add their own properties and methods, thus reusing the common code rather than rewriting it.

Polymorphism

You can think of *polymorphism* as a “capability” of inheritance, although it is actually another OOP concept of its own that is heavily tied to inheritance. When one class is derived from another, the methods and properties are inherited. Polymorphism addresses the problem of modifying one or more of the inherited methods or properties. This prevents having to skip using inheritance simply because one method or property is not compatible.

The formal definition of polymorphism is the capability of classes to provide the same method or property, and the code calling or using it does not have to know which class it belongs to.

Inheritance and Polymorphism Example

Inheritance and polymorphism are probably best explained using an example. This particular example is completely hypothetical and code is not produced, as it is the principle that is important to understand.

An air traffic control system detects incoming planes, and it uses objects to represent all the planes currently in the vicinity of the airport. The programmer of the system creates a class called Plane. This is a class that provides methods and properties that apply to all planes. Examples of such methods would be ClearToLand and ClearForTakeoff. Properties might include the flight number. The programmer may then use inheritance to create a new class called CargoPlane that is derived from the Plane class. In this class a new property, CargoMass, can be added. The programmer could then develop another class called PassengerPlane, which again is derived from the Plane class. A new property, Passengers, could be added to store the number of passengers on board. However, for a passenger plane, a few more steps need to take place when it is cleared for landing: The luggage needs to be offloaded and moved to the correct terminal. Thus, the method ClearToLand could be overloaded (polymorphism) and replaced with one that performs the appropriate operations, while all the other properties and methods of the class remain intact.

Summary

This chapter has given you a very brief introduction to the Web-related technologies that are essential in understanding Web application development—namely, HTML, CSS, and JavaScript. I explained the roles of client and server-side scripting, and presented a short overview of Web servers and server-side programming. Additionally, you took a quick look at the Visual Basic .NET language and the changes that have been made to it, and you learned some of the core VB .NET and .NET Framework concepts. In the next chapter, you’ll be introduced to the Visual Studio .NET IDE and build your very first ASP.NET Web Form.

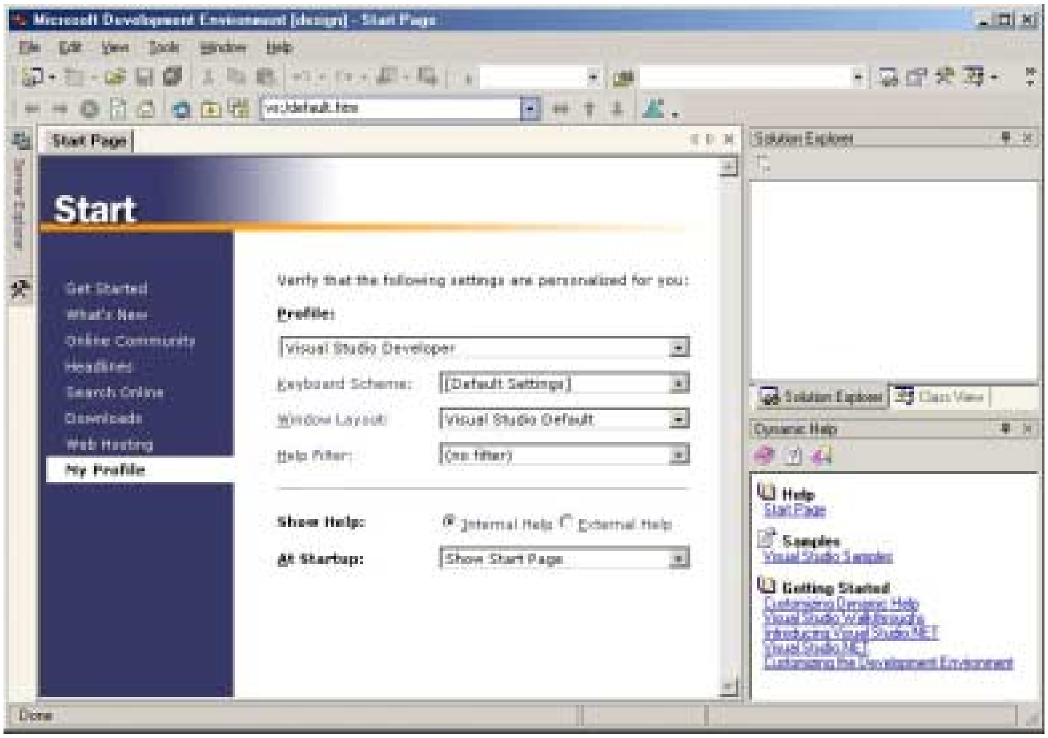
Chapter 4

Your First ASP.NET Web Form

This chapter serves as an introduction and a guide to the Microsoft Visual Studio .NET integrated development environment (IDE), as well as a practical tutorial on getting started with ASP.NET Web Forms.

Welcome to Visual Studio .NET

Visual Studio .NET (VS .NET), although totally revamped to include support for and interaction between multiple languages (as well as a host of new features that will be introduced in this chapter and throughout this book), will look very familiar to users of previous Microsoft development tool products, such as Visual C++, Visual Basic, and Visual InterDev. When you first start VS .NET, you see a screen similar to the one shown in Figure 4-1.



Verify that the following settings are personalized for you:

Profile:

Visual Studio Developer

Keyboard Scheme: [Default Settings]

Window Layout: Visual Studio Default

Help Filter: (no filter)

Show Help: ● Internal Help ○ External Help

At Startup: Show Start Page

Insert 0015f0401.tif

Figure 4-1. The VS .NET IDE at start-up

Probably the most noticeable difference at first glance is the new HTML-formatted welcome area in the center of the screen. This allows you, among other things, to set a “profile” for VS .NET from the list of previous Visual Studio development tools (Visual InterDev, Visual Basic, VC++, and so forth), so that developers using the aforementioned environments will be instantly familiar with the interface and layout of dialog boxes. For those who have not used or are not familiar with any Microsoft development tools, the Visual Basic layout is the one used throughout this book.

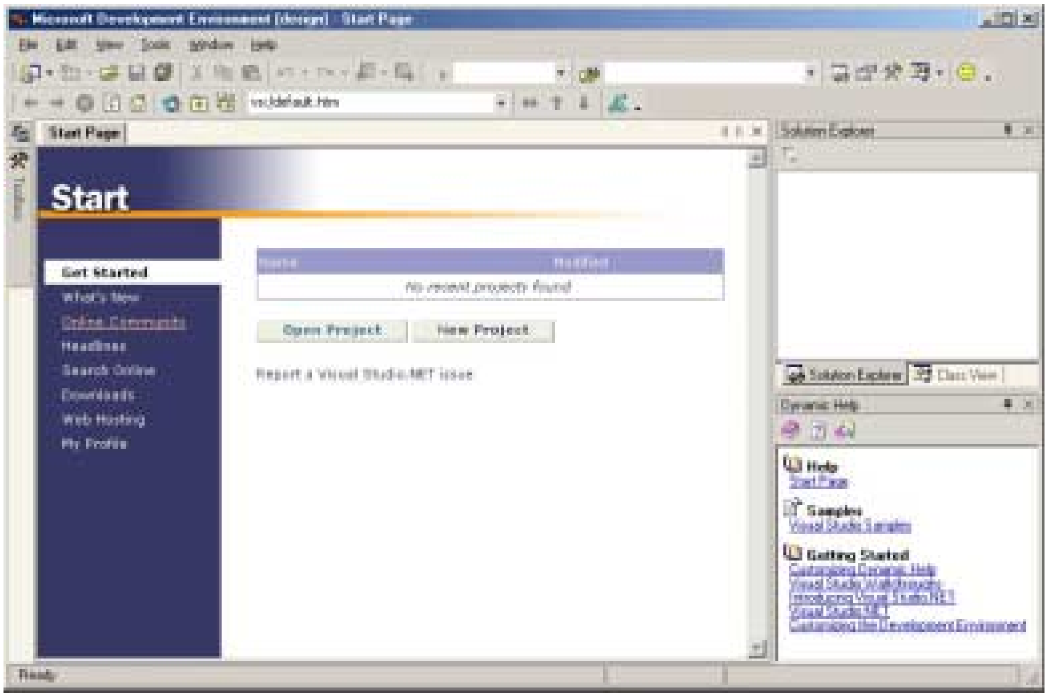
For clarification, the Keyboard Scheme setting refers to the keyboard shortcuts of the IDE, and the Window Layout setting refers to the visibility and location of certain dialog boxes, such as the Properties window and the Solution Explorer. The Help Filter setting's effects will become clearer later, but for now be aware that it sets the help system to filter certain content from the help files. This is because the help files include a myriad of examples in many different languages apart from VB .NET (for instance, JScript .NET, C++, and C#). This filter can block out samples and declarations that are not pertinent.

The choice between integrated and external help is best left as Integrated. However, users who prefer help to be available in a separate window rather than integrated as another tab in the IDE should choose the Outside the IDE option.

The At Startup option defines what should be automatically shown on start-up. The default Visual Studio Start Page, which will be shown later, provides a useful starting point for Visual Studio.

Visual Studio Start Page

What is shown at start-up is actually the My Profile section of the Visual Studio Start Page. Unless you set up it otherwise, all subsequent start-ups of Visual Studio will display the Get Started page of the Visual Studio Start Page, as shown in Figure 4-2.



Insert 0015f0402.tif

Figure 4-2. The VS .NET Get Started page

This page displays all recent solutions (*solutions* are groups of projects in Visual Studio), which you can open, as well as options to create new projects and open existing (but not necessarily recent) ones.

The What's New section details the differences between VS .NET and previous versions of Visual Studio. The sections Online Community, Headlines, Search Online, Downloads, and Web Hosting provide convenient access to various parts of the Microsoft Developer Network (MSDN) at http://msdn.microsoft.com or other online resources.

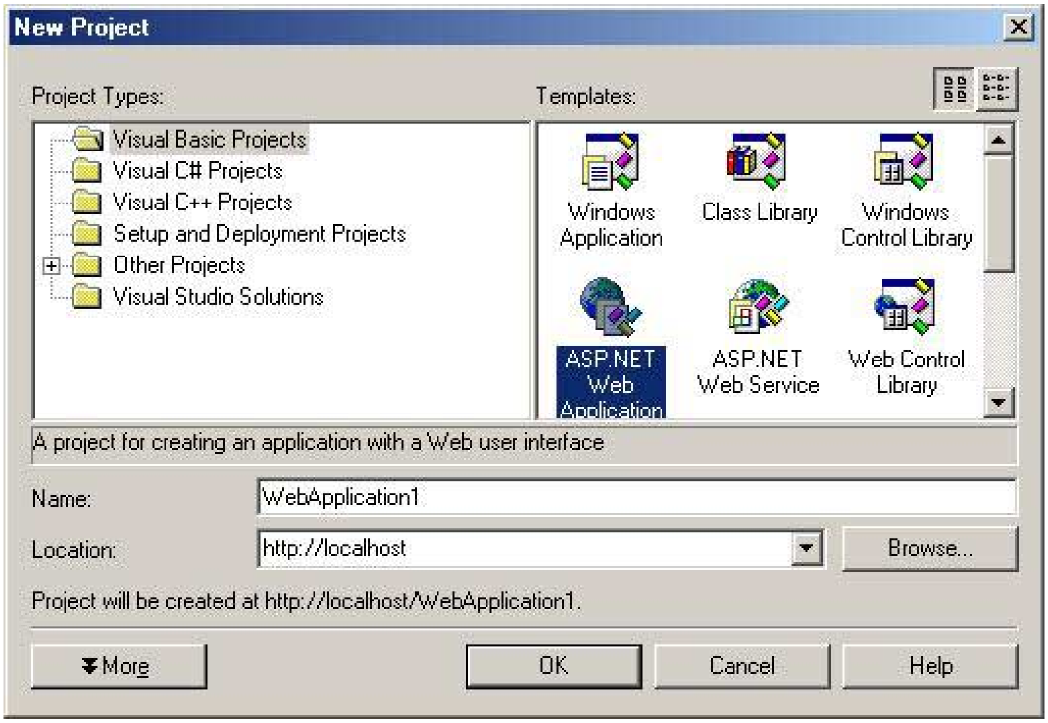
Exploring the IDE

The best way to familiarize yourself with some of the features of VS .NET is to have a hands-on tour. Visual Studio has a very substantial feature set, however, there are many features of Visual Studio that are commonly used and offer a great productivity boost if used effectively. This section aims to introduce those common items.

Starting Up

Perform the following steps to create a new Web project:

1. Start VS .NET.
2. Click New Project in the Get Started section of the Visual Studio Start Page. The New Project dialog box appears, as shown in Figure 4-3.
3. Click the Visual Basic Projects folder in the left pane of the New Project dialog box.
4. Click the ASP.NET Web Application icon in the Templates window.
5. Click OK.



A project for creating an application with a Web user interface

Name: WebApplication1

Location: http://localhost

Project will be created at http://localhost/WebApplication1.

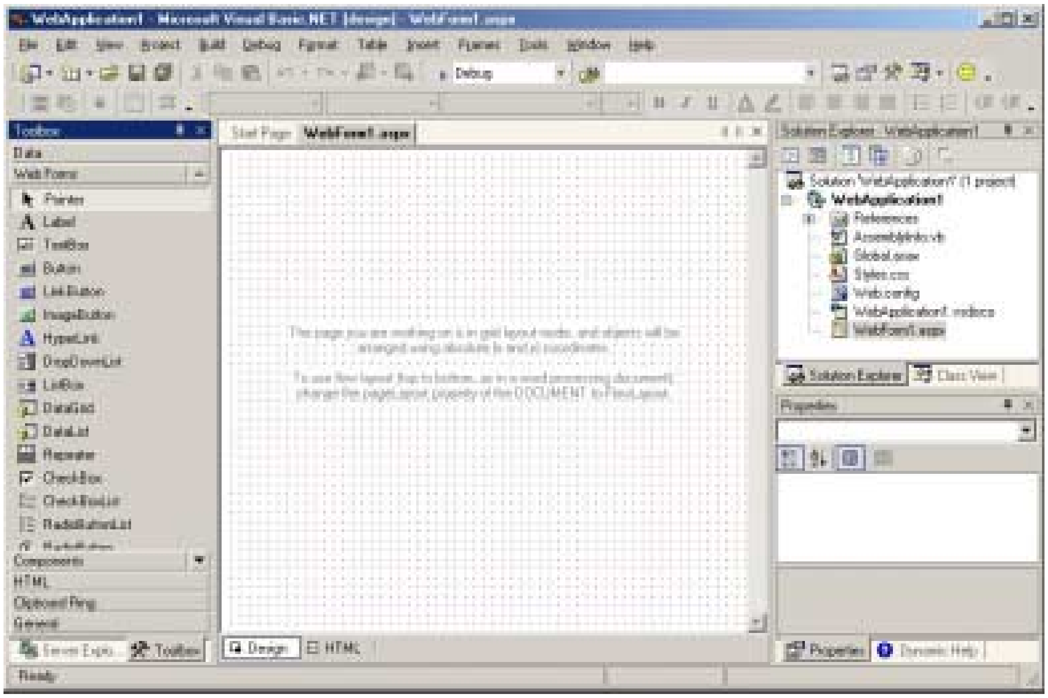
Insert 0015f0403.tif

Figure 4-3. The New Project dialog box

I discuss what Visual Studio does to set up the project in more detail later, but essentially all it does is create a collection of files that form the skeleton of an ASP.NET application and set the application up in IIS.

Design View

Once you've created a new project, Visual Studio should look similar to Figure 4-4, depending on the settings in the My Profile section of the Visual Studio Start Page.



The page you are working on is in grid layout mode, and objects will be arranged using absolute (x and y) coordinates.

To use flow layout (top to bottom, as in a word processing document), change the pageLayout property of the DOCUMENT to FlowLayout.

Insert 0015f0404.tif

Figure 4-4. Visual Studio with the Design view displayed

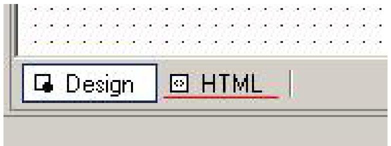
You can think of this page designer as analogous to the Form Designer in traditional VB—it is an empty canvas where you build the user interface (UI) by dragging and dropping components onto it. In this sense it is very much the same as the Form Designer, although there are several differences. Web pages do not have specific boundaries like those that can be given to forms, so that they can't be resized to be larger or smaller. Because Web applications are viewed in Web browsers, the page can be manipulated in any way that Web browsers can manipulate any other Web page. This is a very significant point, and it is one of the more difficult concepts that needs to be properly understood in the tradition between developing traditional Windows applications and Web applications.

The Design view is technically a WYSIWYG editor ("What You See Is What You Get," which holds true for the Form Designer in Visual Basic). A control you place on a form will appear in exactly the same place and have exactly the same dimensions as it will when the application is run. Unfortunately, it can often be the case that what you see in the Web Forms Designer is not what you get when you view the page in the browser. More often that not, the problem is related to the dimensions of the page—the distances used in the page make it look great in Design view, but when you see it in a browser (which will likely have a much larger viewing area), the distances look completely out of proportion. To add insult to injury, there is no OnResize event that is triggered when the browser is resized, so the page cannot be dynamically redrawn appropriately every time the user resizes the browser. For this reason, the use of HTML tables is crucial in maintaining correct distances, as tables allow specific dimensions to be specified for the entire table, as well as rows and columns.

Design considerations aside (as they apply to all Web pages, not only those developed using the Web Forms Designer), Design view offers a fairly powerful and easy-to-use UI designer. All Web Forms in Visual Studio have two files by default: one to hold the content and another to hold the server-side logic code. The Visual Studio designer ensures that appropriate entries are made in both files when controls are added to the Web Form. This is a major timesaver and relieves much annoyance, as it is very easy to forget to add appropriate entries to the logic code file if an entry is made in the UI file manually.

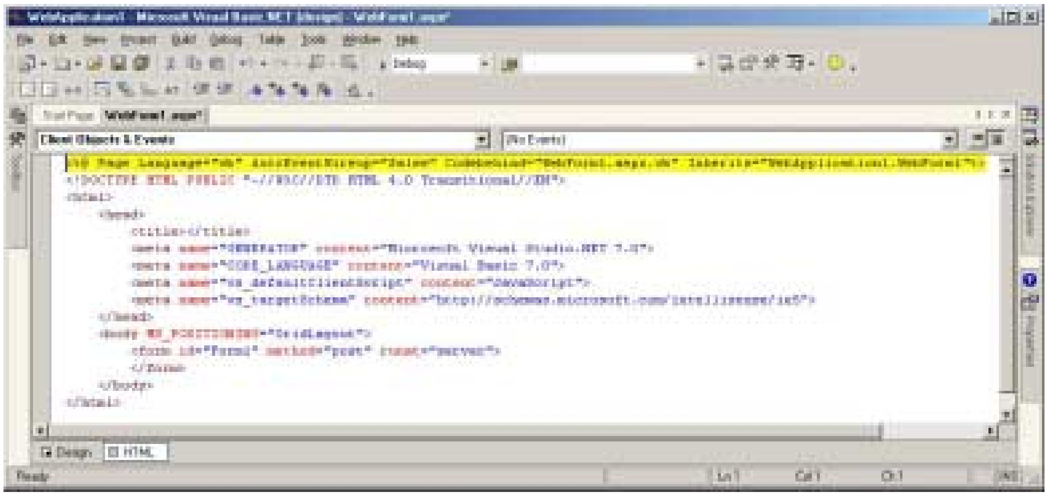
HTML View

The name "HTML view" is not entirely accurate. This view, which you access by clicking the HTML button at the bottom of the Web Form Designer as shown in Figure 4-5, shows the UI code for a Web Form. This code is primarily HTML, although there are several important differences between the UI code and vanilla HTML, which will be clearer later in this chapter and in the next chapter. Figure 4-6 shows the HTML view.



Insert 0015f0405.tif

Figure 4-5. Button to toggle the HTML view


WebApplication1 - Microsoft Visual Basic .NET [design] - WebForm1.aspx*
File Edit View Project Build Debug Table Tools Window Help
Debug
Start Page
WebForm1.aspx*
Client Objects & Events
(No Events)
Design
HTML
Ready
Ln 1
Col 1
Ch 1
INS

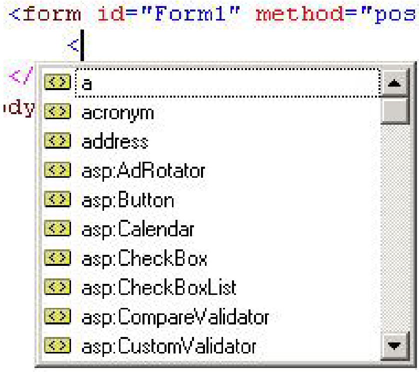
```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb" Inherits="WebApplication1.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
    <head>
        <title></title>
        <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
        <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/intellisense/ie5">
    </head>
    <body MS_POSITIONING="GridLayout">
        <form id="Form1" method="post" runat="server">
        </form>
    </body>
</html>
```

Insert 0015f0406.tif

Figure 4-6. HTML view for a blank Web Form

The idea of manually editing UI code will seem very foreign to developers who have only built Windows applications in a drag-and-drop environment. As I explained in Chapter 3, building Web applications is different and there are certain things that you can only do by going into the HTML source and modifying it directly. The HTML editor in Visual Studio is very powerful and much easier to use than a plain text editor such as Notepad. All syntax is highlighted, which helps in distinguishing tags and code from content. Additionally, the editor incorporates Microsoft's

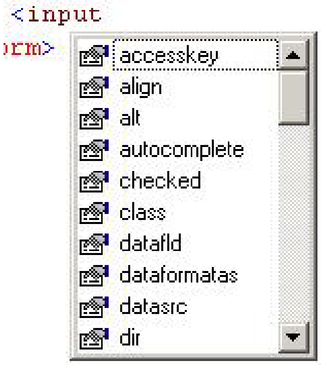
AutoComplete technology, which developers who have used Visual Basic 5.0 and above and Visual InterDev 6.0 will recognize. In HTML view, typing the angled brace (<) that signals the start of a tag will result in a menu popping up that displays tags that can be inserted, as illustrated in Figure 4-7. AutoComplete also provides lists of attributes that you can insert when building a tag. Figure 4-8 shows this feature.



```
<form id="Form1" method="post"
    <
</
dy
```

Insert 0015f0407.tif

Figure 4-7. HTML tag AutoComplete



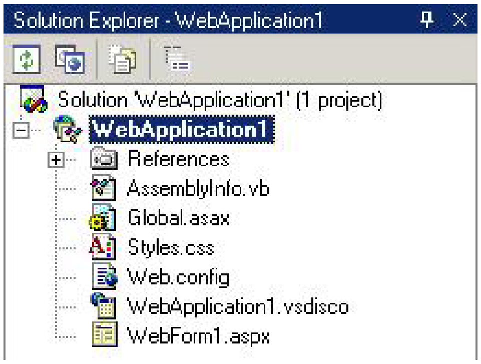
```
<input
rm>
```

Insert 0015f0408.tif

Figure 4-8. HTML attribute AutoComplete

Solution Explorer

The Solution Explorer is a module that allows for solution management. A *solution* in Visual Studio is analogous to a group of projects. However, the Solution Explorer is much more than a simple embedded file manager. It provides the facility for References to other local .NET classes. It also provides the Web References functionality, which allows for consumption of Web Services. Figure 4-9 shows the Solution Explorer for the current project with only one Web Form.



Insert 0015f0409.tif

Figure 4-9. The Solution Explorer

As you can see, there are already numerous files listed in the project: config.web, Global.asax, Styles.css, WebForm1.aspx, and WebApplication1.disco. Table 4-1 outlines the purpose of each file.

Table 4-1. Files Shown in the Solution Explorer When a New Web Project Is Created

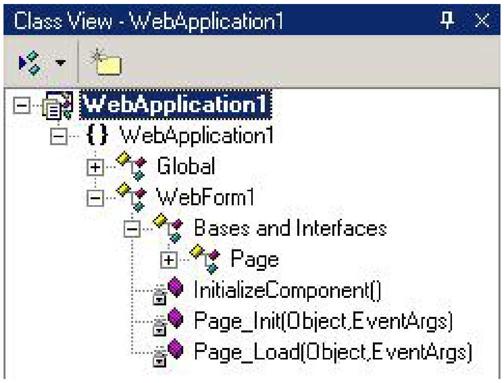
File	**Purpose**
Web.config	Holds configuration information for the entire project (e.g., authentication requirements and so forth).
Global.asax	Provides the ability to add code to event handlers for events such as the start and end of the application, the start and end of a user session, and the start and end of a request.
Styles.css	This is a CSS document that is linked to by all Web Forms and holds the styles for the entire project.
WebForm1.aspx	This is the UI file for the one Web Form that is created automatically when a Web Forms application is created. This is the file that is currently loaded in the Web Forms Designer, either in Design or HTML mode.
WebApplication1.vsdisco	This file holds information about the project for the Dynamic Discovery specification. It deals with the discovery of Web Services exposed by the project, and it is rarely necessary to manually edit the contents of this file.
AssemblyInfo.vb	This Visual Basic file uses attributes to describe the assembly that will be created for this project.

There is actually one more file that has been created, although it is not shown in the Solution Explorer by default. It is the code-behind file for WebForm1.aspx. WebForm1.aspx only holds the UI and references to event handlers for that specific Web Form. The code that drives the page is contained in a file named WebForm1.aspx.vb. This will be shown when the code editor is introduced.

Class View

The Class View is displayed by selecting the Class View option from the View menu. The Class View shows all the classes that apply to the project, including those in the project's References and Web References, as well as classes that the project itself owns (each Web Form is in itself a class).

The Class View is a powerful tool and provides a constructive and useful overview of a solution. Members of classes, such as methods and properties, are represented as child nodes in the Class View treeview, and their respective scopes are shown through the use of icons. Figure 4-10 shows the Class View.



Insert 0015f0410.tif

Figure 4-10. The Class View

The Class View also shows bases and implemented interfaces. *Bases* are largely related to inheritance and refer to the class from which a particular class descends. This information has a variety of uses, such as knowing what type of functionality a class provides based on what the class uses as a starting point to extend upon. The concept of interfaces has not been dealt with, but simply, an *interface* is a template for what a class should provide. If a class implements an interface, it provides all the members and associated functionality that is defined in the interface definition.

Toolbox

The Toolbox in Visual Studio is the primary means of adding components to Web Forms using the drag-and-drop paradigm. Although the Toolbox provides essentially the same functionality it did in previous versions, it has been given a major overhaul.

Although divisions between types of components in the Toolbox was previously available, it was rarely used and not a default feature. VS .NET's Toolbox is split into five sections:

* Web Forms
* HTML
* Data
* Components
* Clipboard Ring
* General

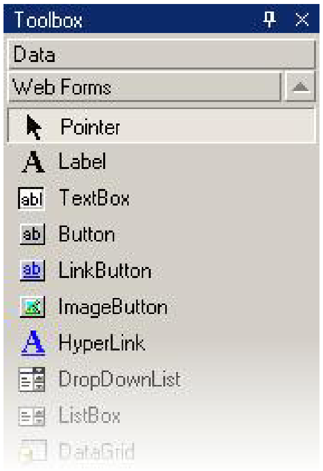
The VS .NET Toolbox, although very similar aesthetically, is actually drastically different under the hood from previous versions. In VB 6.0, the Toolbox was a collection of ActiveX controls from various ActiveX libraries. .NET has its own component model, and COM, COM+, and ActiveX have all been replaced in the .NET Framework.

> **NOTE** Although ActiveX/COM technologies and components are not part of the .NET Framework in any way, tools have been provided to convert COM components to .NET and create COM wrappers for .NET components, so a certain level of interoperability and backward compatibility still exists.

This is particularly pertinent to Windows Forms development, as all controls, although very similar at first appearance, are .NET classes and not controls from ActiveX libraries, such as the Windows Common Controls library. In Web Forms development, the controls available are also all .NET classes.

Web Forms

The Web Forms section of the Toolbox includes visual components that you can place onto Web Forms pages—that is, the components are used for building the UI of the application, along with the items in the HTML tab of the Toolbox. The actual relationships among these components, the .NET classes, and the HTML elements that are generated when the Web Form is viewed in a Web browser is discussed in Chapter 5. Figure 4-11 shows the Toolbox with the Web Forms section selected.



Insert 0015f0411.tif

Figure 4-11. The Toolbox Web Forms section

Table 4-2 outlines the components in this section and provides a short description of their purpose.

Table 4-2. Toolbox Web Forms Section

Name	**Description**
Label	Displays text
TextBox	Text user input
Button	Action user input

LinkButton	Action user input
ImageButton	Action user input, but uses a user-definable image instead of a regular button
Hyperlink	Displays a link to other pages/sites
DropDownList	User input through choices
ListBox	User input through choices
DataGrid	Displays data from databases in grid format
DataList	Displays data from databases in list format
Repeater	Flexible data display mechanism
CheckBox	Boolean user input
CheckBoxList	Boolean user input
RadioButtonList	User input through choices
RadioButton	User input through choices
Image	Displays an image
Panel	Container object for components on a page
PlaceHolder	Container object for components on a page
Calendar	Displays a calendar
AdRotator	Banner advertisement rotation and display
Table	Displays data in table format
RequiredFieldValidator	Ensures a specific field is entered
CompareValidator	Compares the value of a specific field to that of another using a variety of comparisons
RangeValidator	Ensures that a field's value is within a specific range
RegularExpressionValidator	Ensures that a field's value complies with a specific regular expression
CustomValidator	Ensures that a field's value complies with a custom-build validation routine
ValidationSummary	Displays a summary of all validation errors, either inline on the page, in a message box, or both.
Xml	Displays XML or XSL transform results
Literal	Displays text, but styles are not allowed
CrystalReportViewer	Displays a Crystal Report in HTML format

HTML

The HTML section of the Toolbox is the only one that is not strictly tied to .NET. The HTML tab allows certain HTML elements to be inserted into the page. These HTML elements are exactly that—just HTML elements. Simply HTML code is inserted into the page. Because they are not server-side controls, they will not be available to be manipulated by server-side code and should only be used for building noninteractive parts of the UI or for parts where only client-side code is being used. The components in the Web Forms tab are .NET components, and although the net result when a page is viewed in a browser is the same, the Web Controls provide built-in functionality and are accessible via server-side code.

Although the HTML "controls" that are inserted are not server-side controls by default, it is possible to convert them so that they can be accessed on the server-side, as you will be shown in Chapter 5.

Table 4-3 shows the contents of the HTML tab and information pertaining to each item. The Code Inserted column shows the actual HTML code inserted into the Web Form page.

Table 4-3. Code Inserted by HTML Controls

Name	Code Inserted
Label	<div>Label</div>
Button	<input type=button value=Button>
Reset Button	<input type=reset value=Reset>
Submit Button	<input type=submit value=Submit>
Text Field	<input type=text>
Text Area	<textarea rows=2 cols=20></textarea>
File Field	<input type=file>
Password Field	<input type=password>
Checkbox	<input type=Checkbox checked>
Radio Button	<input type=radio>
Hidden	<input type=hidden>
Table	<table cellSpacing=1 width="75%" border=1> <tr><td></td></tr></table>
Panel (flow)Flow Layout Panel	<div></div>
Panel (grid)Grid Layout Panel	<div></div>
Image	<img alt="" src ="">
Listbox	<select size=2><option></option></select>
Dropdown	<select><option></option></select>
Horizontal Rule	<hr width=100% size=1>

Data

As the name suggests, this section of the Toolbox contains components that deal with interfacing with databases. More specifically, these are nonvisual components that provide access to OLE DB–compliant and SQL Server databases through the ADO.NET interface. The components are again all .NET classes. Table 4-4 shows the components available in this tab and their respective functions. All these components are nonvisual and are used either programmatically or for data-binding purposes.

Table 4-4. Toolbox Data Section

Name	Description
DataSet	Adds a typed or untyped dataset to the form
OleDbDataAdapter	Adds a data adapter for OleDb databases to the form, which allows insert, update, delete, and select operations to be performed on a data source

OleDbConnection	Creates a database connection to an OLE DB database
OleDbCommand	Creates a Command object for OLE DB databases for performing specific SQL queries
SqlDataAdapter	Adds a data adapter for SQL Server databases to the form, which allows insert, update, delete, and select operations to be performed on a data source
SqlConnection	Creates a database connection to a SQL Server database
SqlCommand	Creates a Command object for SQL Server databases for performing specific SQL queries
DataView	Controls the properties of a table that controls can bind to

Components

The Components tab is also a collection of nonvisual components. However, these components do not deal with data access. Rather, these components provide important infrastructure functionality that is not data related. Table 4-5 shows the components in this tab and provides a brief description of the components' core infrastructure functionality.

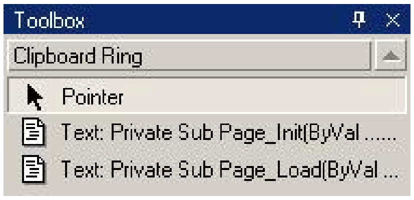
Table 4-5. Toolbox Components Section

Name	**Description**
FileSystemWatcher	Monitors changes to a specified file or directory
EventLog	Manipulates both system and custom event logs
DirectoryEntry	Manipulates and monitors Active Directory entries
DirectorySearcher	Performs queries against Active Directory
MessageQueue	Accesses the operating messaging system
PerformanceCounter	Manipulates and monitors Windows performance counters
Process	Manipulates and monitors Windows processes
ServiceController	Starts, stops, and monitors Windows services
Timer	Time-based trigger functionality
ReportDocument	Manipulates a Crystal Report document

Clipboard Ring

Similar to the functionality of the Microsoft Office 2000 Clipboard toolbar, the Clipboard Ring section of the Toolbox actually has nothing to do with inserting components, but rather has to do with storing multiple items that would normally only be available in the Windows clipboard, one at a time.

The Clipboard Ring is very easy to use and adding items to it as well as "pasting" items from it involve dragging and dropping. To add text to the Clipboard Ring, simply select it and drag it onto the Clipboard Ring window. To "paste" that text, drag the button that was created for it in the Clipboard Ring to where the text must be pasted. A context menu is available for removing items as well as renaming them (through the Delete and Rename Item options, respectively). Figure 4-12 shows the Clipboard Ring with several items added.


Toolbox
Clipboard Ring
Pointer
Text: Private Sub Page_Init(ByVal ......
Text: Private Sub Page_Load(ByVal ...

Insert 0015f0412.tif

Figure 4-12. The Toolbox Clipboard Ring

General

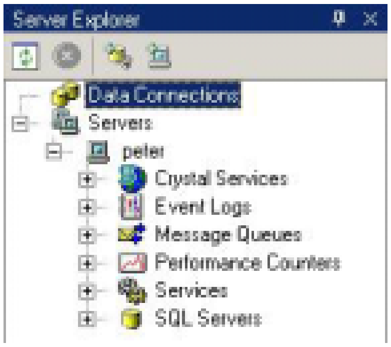
The General section of the Toolbox is empty, save for the ubiquitous Pointer option. This section is dedicated solely to custom third-party components that have been built by either third-party control vendors or development teams.

Server Explorer

The Server Explorer in VS .NET may look very familiar to users of Visual InterDev. Visual InterDev shipped with a feature allowing developers to access and manipulate remote data sources, all in an integrated part of the IDE. The Server Explorer in VS .NET provides this functionality and adds a whole lot of its own. A full list of resources that you can access with the Server Explorer follows (see Figure 4-13):

* Data connections
* Crystal Services
* Event logs
* Message Queues
* Performance counters
* Services
* SQL Server databases

As you can quite clearly see, considerably more than just data access functionality is provided. The Server Explorer virtually eliminates the requirement for multiple copies of Microsoft Management Console (MMC) to be open, monitoring things such as server performance. Most information that developers require about the computing environment is available at, literally, the click of a button.



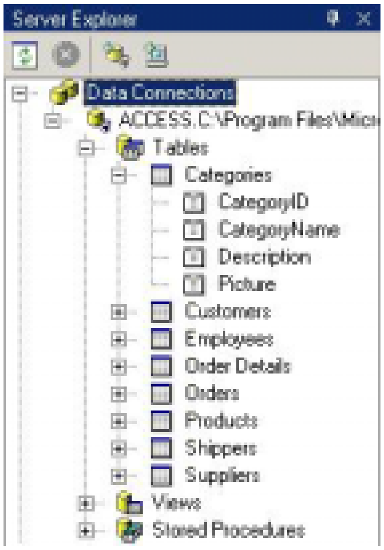
Insert 0015f0413.tif

Figure 4-13. The Server Explorer

Data Connections

Connections to OLE DB data sources, as well connections directly through the SQL managed provider, are shown in the Data Connections treeview of the Server Explorer (see Figure 4-14). From here, you can edit and create tables, manipulate stored procedures, and perform many other common database tasks. Naturally, this facility does not provide all features for all databases, but it does provide a broad variety of functionality that will help you avoid loading separate management applications for most routine development tasks.

This feature and its relationship to database connectivity in Web Forms applications is discussed later in Chapter 8. .



Insert 0015f0414.tif

Figure 4-14. Server Explorer Data Connections section

Crystal Services

Several editions of VS .NET feature the Crystal Reports software, for building both complex Windows Forms and Web-based, data-driven reports. Crystal is an entirely separate product in its own right, and it beyond the scope of this book to cover it, but this integration with VS .NET helps Crystal developers access and administer the reporting software more easily.

Event Logs

Developers who have only been developing applications for and using the Windows 9*x* platform will probably not be familiar with the concept of Event logs. Event logs are essentially a feature of Windows NT/2000 that allows applications to write critical moments in their execution to a central repository of application events, from which administrators can assess the situation of a machine. Events are usually written when something critical happens that may be of use to the systems administrator or developer in troubleshooting. Events such as the start and termination of a critical service are a common example.

The Event Logs tree of the Server Explorer allows developers to view a particular machine’s logs, helping with both application debugging and troubleshooting, as well as ensuring that the application is posting the correct events at the correct time itself. Again, all this functionality, which is roughly equal to that exposed by the MMC event log viewer, is integrated right into Visual Studio.

Message Queues

Message Queues are often an essential part of many enterprise applications and deal with providing guaranteed message delivery (messages normally contain instructions, such as operations to be performed on a database, like adding a row). Message queuing basically involves a series of machines running Microsoft Message Queue, which allows messages to be sent and received. The topic of message queuing to build ultrareliable applications is beyond the scope of this book, but suffice it to say that the Message Queues tree allows developers to easily control and monitor the messages being sent to and received by various machines.

Performance Counters

Windows NT and 2000 sport a very comprehensive array of performance monitoring facilities. By default, graphs that can be plotted include areas such as CPU time, disk access time, and so forth. However, the system is very extensible, and a whole selection of .NET-related counters are installed with the .NET Framework. These include counters in the following categories:

* .NET CLR Exceptions
* .NET CLR Interop
* .NET CLR Jit
* .NET CLR Loading
* .NET CLR Locks
* .NET CLR Memory
* .NET CLR Remoting
* .NET CLR Security
* ASP.NET
* ASP.NET Applications

These monitors are particularly important when testing application scalability, with the counters in the ASP.NET and ASP.NET Applications categories being particularly pertinent. Additionally, it is possible to add your own counters to the Windows system in Web applications. These could be used to monitor a multitude of items—for example, currently logged-in users. The Performance Counters section of the Server Explorer allows for monitoring of these counters from within VS .NET, as opposed to having to load the appropriate MMC snap-in in a separate window.

Services

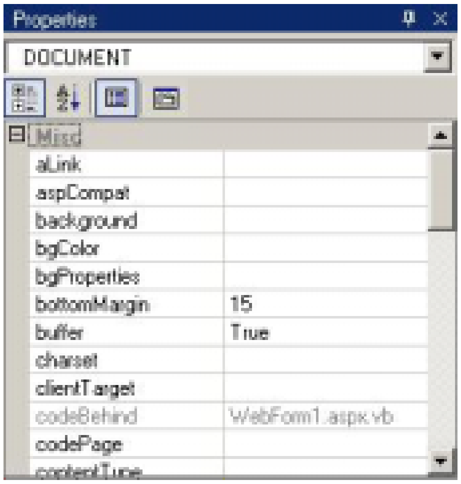
This node lists all the services available on the machine, along with their respective status (started/stopped). Checking running services is often a common place to start non–programmatic-related troubleshooting. Items such as the SMTP service are often critical to an application, and the ability to quickly and easily check on and modify the status of all the services is another useful feature exhibited by the Server Explorer.

SQL Servers

If SQL Server is installed on the target machine, all the databases available on it are accessible through this node. You can use the Visual Studio data tools, which are integrated database development tools inside Visual Studio, to modify most aspects of the databases, including tables and stored procedures. This functionality is very useful when developing database applications, as it alleviates the requirement to use the SQL Server Enterprise Manger or Query Analyzer (or some such generic SQL query application), as most of the required common functionality is integrated directly into VS .NET.

Properties Window

Visual Basic developers will be immediately familiar with the Properties window (see Figure 4-15). It is used to modify properties of visual components on forms and, more pertinent to Web Forms, modify properties of HTML Controls and Web Controls that are placed on Web Forms. However, its functionality is not limited to the Design view of Web Forms, and the Properties window is still fully functionality when in HTML view. However, it cannot be used in code view, when VB .NET code is being edited.



aLink	
aspCompat	
background	
bgColor	
bgProperties	
bottomMargin	15
buffer	True
charset	
clientTarget	
codeBehind	WebForm1.aspx.vb
codePage	
contentType	

Insert 0015f0415.tif

Figure 4-15. The Properties window

The Properties window changes dynamically according to the selected control, as different types of controls have different properties. Naturally, the state of the properties is maintained. The following exercise demonstrates the ease of use and power of the Properties window:

1. Either start a new Web application or use the one already created at the start of the chapter if it is still open.
2. Drag a Button control from the Web Forms section of the Toolbox onto the Web Form in Design view.
3. Make sure that the button is selected (if it is, six black handles will appear around it). If it is not, click it once.
4. Scroll down to the Text property in the Properties window, and modify the property by clicking in the cell adjacent to the property name that says “Button” and editing that text to read “Hello world!”. Notice that when that edit field loses focus (when you click out of it or press Enter), the Design view immediately shows the modification.
5. Scroll down to the BackColor property and choose a color from the drop-down list that appears for that property.

This quick introduction to the Properties window demonstrates two of the three types of properties that have slightly different representations in the Properties window. The first is the basic property where a value is entered, with the second being one where a choice is given (with color values and Booleans being the most common property values where this happens). The third is the set, or collection of properties within one central property. The most common example of this is the Font property, which does not have any value itself, but contains other properties to describe the font, such as size, name, and stylistic functionality (bold, underline, italic, and strikeout). You can easily recognize properties that are sets in the Properties window, as they will have a small plus sign (+) next to them, denoting that you can expand them to expose subproperties.

IDE Summary

There is much more to the Visual Studio IDE than I’ve covered in this chapter, but you have learned about the basic dialog boxes that you will most commonly use in UI design. As you can clearly see already, Visual Studio is far from a souped-up text editor for writing code with some cheap WYSIWYG tools thrown in. It is a powerful, rich environment for designing Web user interfaces. The fully integrated coding and debugging tools will be introduced shortly to provide the full picture of what Visual Studio is and why it is such a productive development environment.

Building a Simple Web Form

This section is intended to demonstrate how to build a simple Web Form and also to explain some of how ASP.NET Web Forms work from the inside out. Along the way, you'll be introduced to several new elements of the Visual Studio IDE.

Starting Up

Creating a new Web Forms application, as described near the beginning of this chapter, is where almost all new ASP.NET applications begin. You will learn shortly what actually happens when a new application is created, along with the uses of the myriad of files that are created. For now, create a new Web Forms project, and call the project MyFirstWebApp instead of WebApplication1. Simply put, Visual Studio now creates a comprehensive framework for the application that can easily be built upon, along with one Web Form to start from.

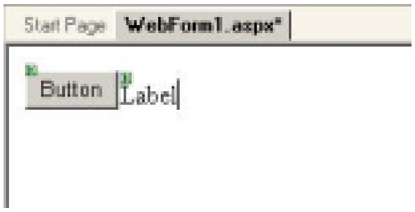
Designing the Web Form

Designing Web Forms in Design view is literally as easy as drag and drop. To add controls to the form, simply drag them from the Toolbox and onto the form.

> **NOTE** There are two modes in which you can create Web Forms: Grid Layout mode or Flow Layout mode. The difference between the two is essentially that in Grid Layout mode, the form design is performed almost identically to form design in Visual Basic applications. When a component is dropped onto the form, it snaps to the nearest points on a grid, giving the UI designer almost 100 percent precision over the positioning of controls on the form. Flow Layout mode follows a word processor style of design. The position of controls is relative to the other controls on the page. In this book, Flow Layout mode is used unless otherwise specified. I present instructions for modifying the layout mode of the form shortly.

You can insert all content on the form that does not need to be programmatically accessed by simply typing on the form, as you would with a word processor. However, any content that needs to be programmatically accessed must be in the form of Web or HTML Controls, which you insert by dragging and dropping from the Toolbox.

To start with your form, add one Button and one Label Web Control to the form. However, before doing this, you need to change the layout mode of the form. To do this, choose DOCUMENT in the Properties window and change the pageLayout property to FlowLayout. After you add the two controls, the Form Designer should look similar to Figure 4-16.



Insert 0015f0416.tif

Figure 4-16. Web Form in Flow Layout mode with Button and Label Web Controls added

The Generated “HTML”

After you’ve added the Web Controls, switch to the HTML view of the Web Form. The code displayed should look similar, if not identical, to Listing 4-1.

Listing 4-1.

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb"
 Inherits="MyFirstWebApp.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
 <HEAD>
  <title></title>
  <meta content="Microsoft Visual Studio.NET 7.0"
name="GENERATOR">
  <meta content="Visual Basic 7.0" name="CODE_LANGUAGE">
  <meta content="JavaScript" name="vs_defaultClientScript">
  <meta content=http://schemas.microsoft.com/intellisense/ie5
 name="vs_targetSchema">
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:Button id="Button1" runat="server" Text="Button"></asp:Button>
   <asp:Label id="Label1" runat="server">Label</asp:Label>
  </form>
 </body>
</HTML>
```

There are three points to note about this “HTML” code that has been generated. The first is the runat attribute of the form element. The HTML standard does not include a runat attribute for the form element, but because this file is going to be processed before it is actually sent to the client browser, ASP.NET files can contain syntax that is not strictly HTML. In this case, the runat attribute exists so that ASP.NET knows to process this form for things that need to happen on the server side before the file is sent to the client.

The second snippet of note is the <asp:button> “element.” Again, HTML does not contain such an element. This is actually the ASP.NET UI syntax for a Web Control, and it signals for ASP.NET to process this element on the server side. Note that this “element” also contains a runat attribute.

Third is the <asp:label> element. Identical to the previous point, this is the syntax used in ASP.NET to denote a Web Control—this time, a label.

Viewing and Editing the Server-Side Code

Visual Basic and client-side JavaScript programmers will certainly be familiar with the concept of events and event-driven programming. Most ASP programmers will probably have come across the event-driven programming paradigm as well. It is very important that you understand the event-driven programming model, as all ASP.NET applications revolve around this principle.

For the uninitiated, event-driven programming revolves around building code that is executed according to the actions the user performs (for our purposes, at least—event-driven programming does actually encompass a significant amount more than this). The programming language and infrastructure performs all the plumbing of capturing user actions and providing the appropriate conditions, and most of what the programmer is required to do is write the code that needs to be executed when a particular event occurs. An example of an event is the clicking of a button.

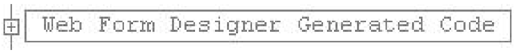
Event handlers are the subroutines that are called when an event occurs. Visual Studio can generate blank event-handler blocks and link them (hook them up) to the appropriate objects. To get Visual Studio to do this, double-click the Button Web Control on the form. This will hook up a click event for the button. Visual Studio will also automatically switch to the code (not HTML) view, which displays all the server-side code for the Web Form. Listing 4-2 shows the code that is currently being displayed.

Listing 4-2.

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

    End Sub

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As _
 System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub
```

```
#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click

    End Sub
End Class
```

The code between #Region and #End Region may not be visible and replaced by Figure 4-17.



Web Form Designer Generated Code

Insert 0015f0417.tif

Figure 4-17. Expandable code region marker

This marker indicates there is code that is not being shown. Clicking the plus sign (+) will reveal the code. This functionality operates very similarly to treeview controls, but it contains code rather than nodes. Thus, after expanding the code by clicking the plus sign, the sign will change to a minus sign (–), and clicking the minus sign will hide the code again. Notice that all Subs also have this functionality, which at first may seem fairly pointless, but is actually very useful when dealing with large blocks of code. For Visual Basic developers, this replaces the Sub View mode, where only one subroutine could be viewed at a time.

Dissecting the Pregenerated Code

With the exception of the control definitions and Button1 click event handler, all Web Forms created using Visual Studio will by default have the server-side code shown in Listing 4-2.

The Public Class WebForm1 statement defines the start of a class for the Web Form. Note the End Class statement at the end of the code.

The Inherits statement is used to instruct the compiler to inherit all the inheritable members (properties, methods, and events) of the System.Web.UI.Page class.

For those unfamiliar with inheritance, this essentially means that the WebForm1 class contains everything a Page class does, and then adds its own members. This is important, as the Page class contains many members that are critical to interacting with both the server and the client, and forms the base for all ASP.NET Web Forms.

NOTE Chapter 3 includes a slightly more in-depth discussion of inheritance.

The next two statements (repeated in the code that follows) are the control declarations for the two Web Controls that were added to the form. The reason these declarations need to be included in the server-side code when they already exist in the UI code is so that they are accessible from within the class and can have event handlers attached to them.

```
Protected WithEvents Button1 As System.Web.UI.WebControls.Button
Protected WithEvents Label1 As System.Web.UI.WebControls.Label
```

Next is the code region. *Code regions* are used for grouping together related, adjacent pieces of code. In Visual Studio this results in a collapsible region being created.

The private InitializeComponent sub is used for any component initialization that needs to occur. This does not currently contain any code, but this event handler is normally used when nonvisible controls are added to the page.

The Page_Init Sub is an event handler for the Page's Init event. Event handlers generally take the form of ObjectName_EventName. In this event handler, the InitializeComponent routine is called. All code that may need to be included in this event handler should follow that call.

The Page Init event is triggered when a page is loaded, followed by the Load event. The Page_Load method is the event following the Init event.

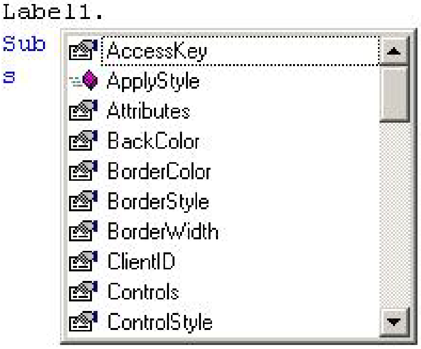
Finally, the Button1_Click event handler is what was created when the Button on the form was double-clicked. This method is called when Button1 is clicked.

Adding Code to the Event Handler

Adding code to the event handler is, well, exactly as you would expect it to be. Place the cursor in the space between Protected Sub Button1_Click(…) and End Sub and type the appropriate code in. For now, all the code will do is change the text of the label (Label1) to, you guessed it, "Hello world!". Type in the following code, and pause after you type the period:

```
Label1.Text = "Hello world!"
```

Pausing after typing the period should result in a list box being displayed, as shown in Figure 4-18.



```
Label1.
Sub
s
```

Insert 0015f0418.tif

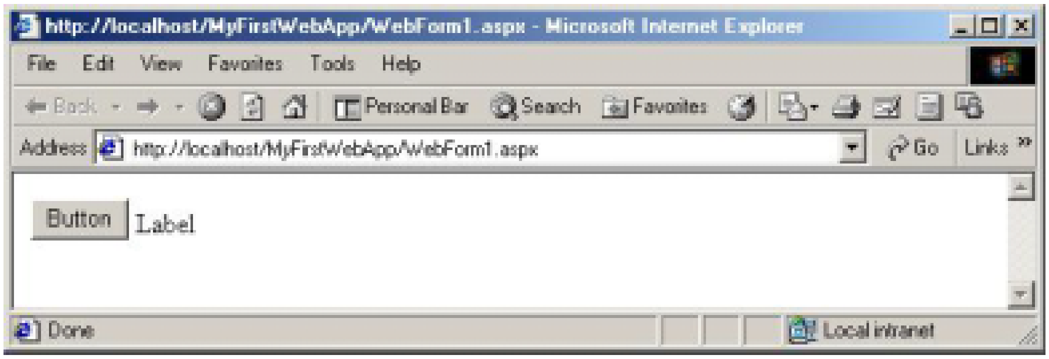
Figure 4-18. IntelliSense statement completion

The period in VB .NET is the member selection operator. Visual Studio, realizing that a member of the Label1 object is about to be referenced, preemptively displays a list of choices of all the members of Label1 that can currently be accessed. If you type *T*, the list scrolls down to all the choices beginning with *T*. This feature is particularly useful when you know what a method name starts with but are not quite sure how to spell it. It prevents the trouble of loading help and looking up the member in question or, if the object is a custom-built class, opening the class' code to check. Complete the code as shown previously. This code assigns the string "Hello world!" to the Text property of Label1.

Running the Application

Run the application either by using the menu command Debug-Start or by pressing F5. The IDE will change form slightly and go into debug mode. Basically, this means that the Form Designer and several other design-time windows are hidden if they were visible, and debug tool windows may be opened.

Internet Explorer should appear, showing a page similar to that shown in Figure 4-19.


http://localhost/MyFirstWebApp/WebForm1.aspx - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Personal Bar
Search
Favorites
Address
http://localhost/MyFirstWebApp/WebForm1.aspx
Go
Links
Button
Label
Done
Local intranet

Insert 0015f0419.tif

Figure 4-19. "Hello world!" application running in Internet Explorer

After you click Button1, there will be a slight pause, and the text Label will be changed to "Hello world!". Before you close Internet Explorer, note the URL of the page. It should be in the form of http://localhost/WebApplication1/WebForm1.aspx. Closing Internet Explorer will return Visual Studio to Design mode.

Behind the Scenes

For what is a very small and trivial example, there is actually a large amount happening behind the scenes, as it were, both in terms of creating the application and running it.

Application Creation

Every time a new Web application is created in Visual Studio, several operations are performed. First, a new directory is created in the IIS default site's physical root. By default the physical root is c:\inetpub\wwwroot. This directory is of the same name as the name chosen for the Web application in the New Project dialog box. Second, several files are created that form the basis of the application. These include such files as global.asax, web.config, styles.css, as well as one Web Form file set with the same name as the application (each Web Form consists of two files—one aspx file containing the UI code, and one aspx.vb file containing the server-side code). Third, Visual Studio solution and Visual Studio project files are created and placed in the appropriate directories. Finally, a virtual directory is set up in IIS.

> **NOTE** A *virtual directory* in IIS allows directories other than those in the physical Web root to be accessed. For example, if the physical Web root was c:\inetpub\wwwroot, and c:\somesite needed to be accessed, and that folder could not be moved, you could set up a virtual directory so that users who visited http://machinename/virtualdirname would actually receive content from c:\somesite. However, Visual Studio sets up a virtual directory for each application for a different reason—in order for events from IIS to be handled by ASP.NET on a per-application basis, the application must be in a virtual directory.

Application Execution

ASP.NET is a compiled environment, but it alleviates one of the key annoyances of other compiled environments. The developer never has to compile the files him- or herself—all compilation is performed automatically by ASP.NET. Additionally, when a compilation occurs, only the files that have been changed need to be compiled, not the entire application. The following list details the compilation procedure for a totally new ASP.NET application:

1. An aspx file in http://machinename/someapp is hit (someone visited it).
2. ASP.NET goes through the c:\inetpub\wwwroot\someapp directory (assuming c:\inetpub\wwwroot is the physical root path) and compiles all the ASP.NET files in the directory.
3. The originally requested file is served up to the user.
4. All subsequent hits to ASP.NET files in that directory are served immediately (no need for recompilation).

Then assume that somefile.aspx is modified. Here is the next hypothetical sequence of events:

1. somefile.aspx is hit.
2. ASP.NET realizes that the files in the directory have already been compiled and this is not a first hit, so it only recompiles somefile.aspx.
3. somefile.aspx is served up to the user.
4. All subsequent hits to somefile.aspx are served immediately.

In summary, ASP.NET is intelligent in its compilation policy and performs all compilations without any user intervention required, thus combining the speed of compiled environments with the flexibility and ease of use of interpreted environments.

Summary

This chapter has given you a brief introduction to the Visual Studio .NET environment and has guided you through creating a very simple Web application. Before moving on, take some time to familiarize yourself more with the Visual Studio environment, and in particular, try out the different Web and HTML Controls and modify their properties through the Properties window and programmatically.

The next chapter details programming the common Web and HTML controls.

Chapter 5

Programming Web Forms with the Common Web and HTML Server Controls

After last chapter, “Creating you first Web Form”, you’re probably still pretty confused as to exactly how everything in Visual Studio.NET all “fits together” so that you can create a large, useful web application. This is the chapter that shows you how to use all of the most commonly-used Web and HTML Server controls in the Toolbox, how to make them interact with each other, and in the process demonstrate some important concepts of .NET and Visual Basic.NET programming that apply not only to the controls shown in this chapter, but also in a much wider scope, such as collections, enumerations, events and much more.

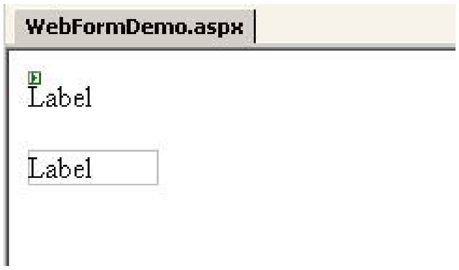
Web Server Controls vs. HTML Server Controls

One of the first possible points of confusion when learning to build web forms is “what is the difference between a Web Server Control and an HTML Server Control?” This probably stems largely from the fact that most of the Web and HTML Server Controls don’t differ much – for example, the Label Web Server Control operates almost identically to the Label HTML Server Control. However, there are some subtle differences between the two.

> Although the official names are “Web Server Controls” and “HTML Server Controls”, they will be referred to as “Web Controls” and “HTML Controls” throughout the rest of the chapter, to help the chapter read more easily.

Web Controls are “server-side” by default; HTML Controls aren’t

When you drag-and-drop a Web Control from the Toolbox onto a Web Form, Visual Studio.NET automatically creates all the necessary code (in both the user interface, and code-behind files) for you to be able to programmatically access and manipulate that control. However, when you drag-and-drop an HTML “Control” onto the form from the Toolbox, all Visual Studio.NET in fact does is insert a piece of plain HTML into the user interface file. Strictly speaking, the “control” that is inserted would not actually be an “HTML Control” yet. To demonstrate this, Figure 5.1 shows a new Web Form with two Labels added – the top one is a Web Control and the bottom one is an HTML “Control”, both dropped from the Toolbox. Listing 5.1 shows the relevant HTML that VS.NET has generated for the page.

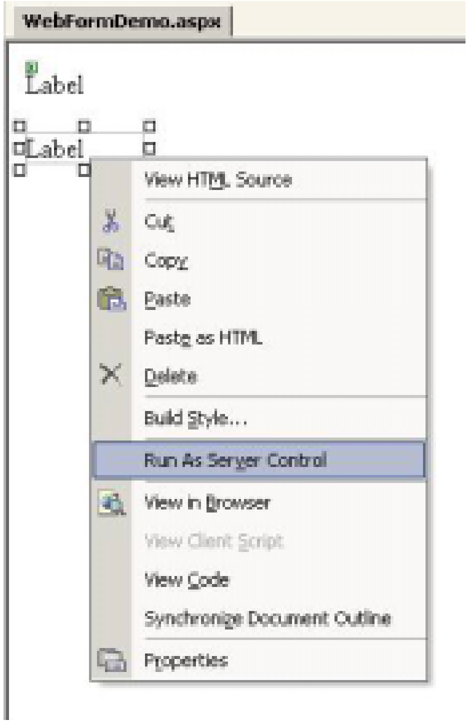


[Figure 5.1]

Listing 5.1

```
<asp:Label id=Label1 runat="server">Label</asp:Label>


<DIV id=DIV1 style="DISPLAY: inline; WIDTH: 70px; HEIGHT: 15px"
ms_positioning="FlowLayout">Label</DIV>
```

For starters, the two entries in the .aspx file look vastly different! Secondly, and more importantly, the second label doesn't have a green "play" icon on its top-left corner, like the Web Control Label does. This icon indicates that the control it is on is a "server-side control", with the effect that it can be manipulated using server-side ASP.NET code. All server-side controls must include one attribute in their declaration tags – "runat". This attribute must have the value "server". When ASP.NET is parsing a page and it reaches a tag that has this attribute on it, it knows that it must make the control available to server-side code. Notice that the Web Control Label has a runat attribute, but the declaration for the HTML Control doesn't. This is because HTML "Controls" aren't actually server-side controls by default. However, HTML controls can obviously be made to be accessible on the server side by one of two methods – you can either manually add the runat attribute to the control's tag in the HTML view, or in the designer, you can right click on the control, and choose the "Run As Server Control" option from the context menu, as shown in figure 5.2.



[Figure 5.2]

Once this option is toggled, the HTML for the control will be modified to include the runat attribute, and the necessary entries will be made in the code-behind (.aspx.vb) file.

Web Controls use a non-HTML syntax; HTML Controls use pure HTML

HTML Controls can simply be inserted to be non-server side, static HTML, and so the code that represents them in the UI (.aspx) file is actual valid HTML. However, Web Controls cannot be non-server side, and when the page is loaded, they will always be processed on the server-side, and can have the actual HTML to display them generated at this stage. This makes Web Controls more flexible than HTML Controls because HTML Controls have to be represented by valid HTML code, whereas Web Controls do not, and have their HTML generated at "run time". This will become evident later, and is the reason that some controls exist only as Web Controls whereas others can be inserted as both HTML and Web Controls.

> Controls that can be inserted only as Web Controls (such as the Calendar control) are made up as a combination of HTML elements, and do not have a direct HTML equivalent, so could not exist as an HTML Control.

Web Controls are represented in the UI code in a custom format that is not valid HTML code. This will naturally be converted to HTML when the page is loaded, but when building Web Forms it is important to understand this.

In the first difference "Web Controls are "server-side" by default; HTML Controls aren't", in Listing 5.1, the different UI code for two labels was shown – the first was for a Label Web Control, and the second for an HTML Control label. Here is the code for two server-side Labels:

```
<asp:Label id=Label1 runat="server">Label</asp:Label>


<DIV id=DIV1 style="DISPLAY: inline; WIDTH: 70px; HEIGHT: 15px"
 ms_positioning="FlowLayout" runat="server">Label</DIV>
```

There is obviously no HTML tag "<asp:Label>", which represents the Web Control, however, the "DIV" or "division" tag is a legitimate HTML tag, and is used to represent a Label HTML Control. All Web Controls' tags begin with the prefix "asp:" Another inherent benefit of the Web Controls representation is that it is much simpler to recognise Web Controls in the UI code, since they use their own simplified syntax. Take the Label control for example – it's much easier to see that <asp:Label id=Label1> is the Label1 control than it is to look for <DIV id=Label1> since the <DIV> tag will probably be used in the page as both the representation for Label HTML Controls and also as static HTML.

Web Controls can represent any control; HTML Controls must have a direct HTML element equivalent

This was touched on in the previous section, but possibly the most important difference between Web Controls and HTML Controls, derived from the fact that Web Controls use their own syntax for representation whereas HTML Controls must use actual HTML code, is that Web Controls can represent any type of control but HTML Controls must be represented by one specific HTML element (tag) with a "runat" attribute.

Of course Web Controls must generate HTML code at run-time, so they are also limited in that they can only do as much as is possible by combining many different HTML elements at run-time to form "controls". For example, the Calendar Web Control, when rendered at run-time, is actually made up using many HTML elements – a table, table rows, table data cells and so forth. However, since there is no actual <Calendar> HTML element, there is no Calendar HTML Control. For this reason, the Web Controls "suite" of controls is much more comprehensive than the set of HTML Controls.

> This may lead you to believe that the Web Controls set of controls includes everything the HTML Controls set does, and then some, and this is largely true – for the most part, there is an equivalent Web Control for all the HTML Controls, all for but two exceptions – there is no file upload Web Control, and if you'd like to let the user upload files to your application, you will need to use the HtmlInputFile HTML Control. There is also no "hidden input" Web control, such as the HtmlInputHidden HTML Control.

Differing object members and functionality

Although there are many similarities between "equivalent" Web and HTML Controls (such as the two dropdowns – the DropDownList control and the HtmlSelect control), there may also be several differences in the members (i.e. properties, events and methods) that are exposed by the respective controls for manipulation on the server side, or even differences in functionality that is exhibited or possible with one control, but not with another. This is a very important factor, and is often decisive when choosing whether to use an HTML or Web Control.

Which to use – Web Controls or HTML Controls?

With many basic controls (such as Labels, TextBoxes etc) being provided as both Web and HTML Controls, the question of which variety should be used, and in what situations, is inevitable. For the most part, the decision is based on personal preference more than actual concrete reasoning, however, although many of the HTML and Web Controls are very similar, there are no two that are identical in form or function, so it is important that you properly understand the differences before choosing to use one or the other. This chapter will show you how to use both, although the rest of the book errs on the side of Web Controls for most examples for several reasons:

* Web Controls generally provide greater functionality than their HTML Control counterparts

■ * Web Controls often have greater Visual Studio integration when designing pages than HTML Controls

* Web Controls have more natural property names than those of HTML Controls, which are specifically tied to the HTML attributes or elements that they affect

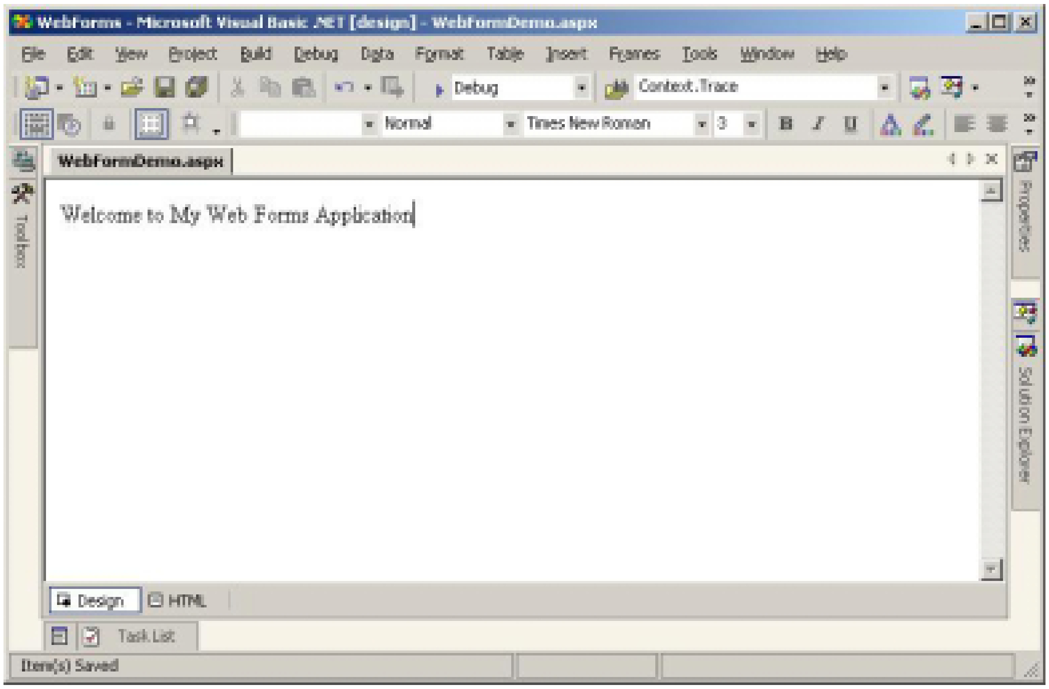
It is important to note that the choice to use one over the other is still largely a matter of personal preference, and you should decide for yourself which control set you'd prefer to use once you've learned the basics of both. It is obviously important that you do know how to use both, even if you decide that in your applications you will use only one or the other, so that you can understand code examples and the like that may use a different method from you.

Learning to use the Controls

In this section of the chapter, we're going to go through each type of control, and demonstrate the use of both the HTML and Web Control varieties, showing all the basic features and possible uses for the control. As each control is introduced, the demonstrations will likely make use of controls that have already been shown to further your perspective of where are how the different controls can interact with each other to make interactive and useful Web Forms.

The Label Controls

With almost all modern visual programming environments, there is a ubiquitous set of controls that makes an appearance. This "set" of controls will often include controls such as buttons, dropdowns, radio buttons and checkboxes, which are all used for user interaction, but the one ever-present control is the Label, which is used not for direct user interaction, but for information display. ASP.NET's HTML and Web Label controls are for this exact purpose – displaying information to the user. However, there is a subtle twist in the usage of the Label control that doesn't exist in the Windows (Forms) programming world – the ASP.NET Label (of either the HTML or Web Control variety) should only be used when the data that is display will be dynamically updated (i.e. if the server-side code is going to manipulate the Label's display). This is because the "form" that is being built is not a Windows Form, but a web page, and is inherently designed for data and information display. Figure 5.3 shows how you can enter data to be displayed on your Web Form without using any specific controls by just typing in the designer.



Welcome to My Web Forms Application

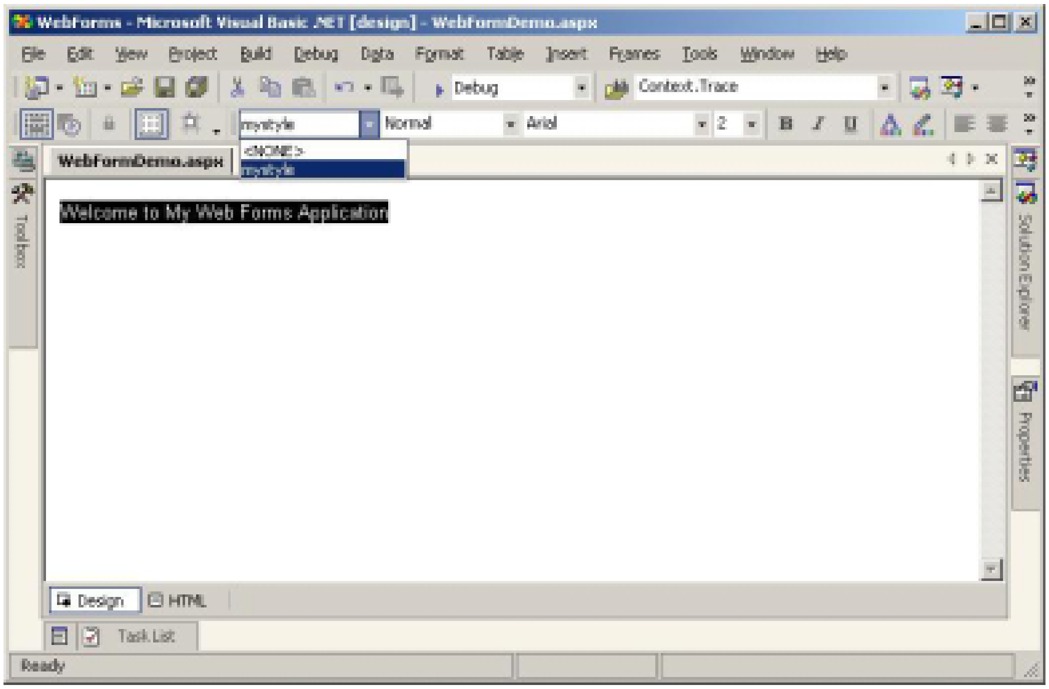
[Figure 5.3]

This text can be manipulated not by using properties (because it's not in a control), but by using the "Formatting Toolbar", which allows basic manipulations such as font name, color and size, along with the ability to make text bold, italic or underlined.

> The "style" dropdown in the Formatting Toolbar can be very useful for applying CSS styles to text (and other page content). To demonstrate this, insert the following code into the <head> section of a UI file in the HTML View:

```
<style>
 .mystyle
 {
   font-family:Arial;
   font-size:10pt;
 }
</style>
```

Switch back to Design View and select the text you want to apply the style to. When you choose an option from the "style" dropdown in the Formatting Toolbar, the style will be applied to the text, as shown in Figure 5.4:

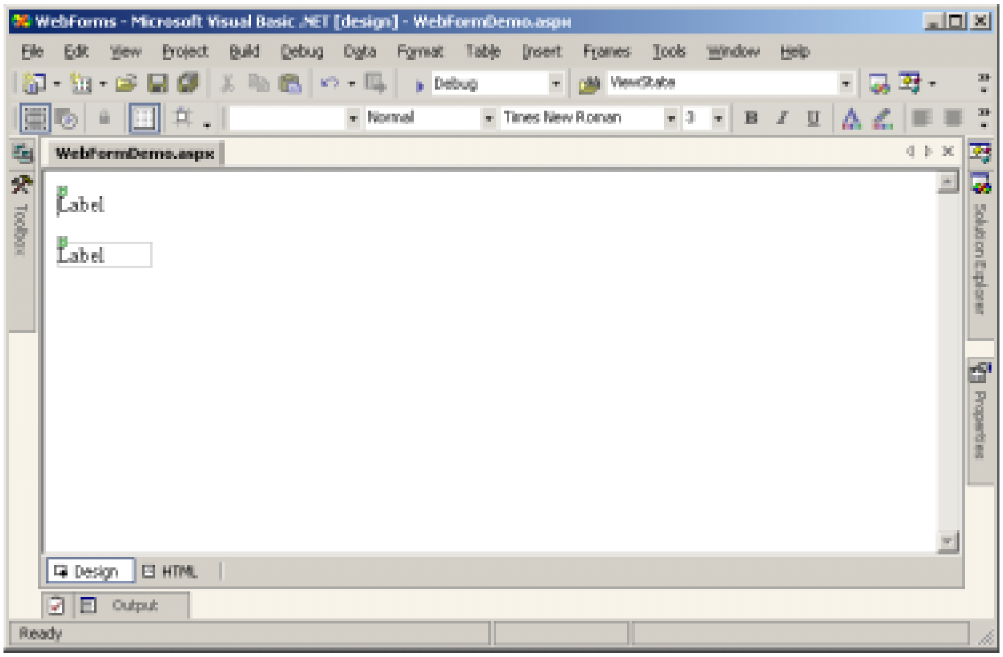

WebForms - Microsoft Visual Basic .NET [design] - WebFormDemo.aspx
File Edit View Project Build Debug Data Format Table Insert Frames Tools Window Help
Debug
Context.Trace
mystyle
Normal
Arial
2
<NONE>
mystyle
WebFormDemo.aspx
Toolbox
Welcome to My Web Forms Application
Solution Explorer
Properties
Design
HTML
Task List
Ready

[Figure 5.4]

With the ability to display formatted text without a Label control, the inclusion of it may seem worthless, until you actually need to modify the text (or the formatting) at run time. Text that is inserted at design time by simply typing in the designer cannot be modified at run-time (at least not easily), which is obviously a major issue, or would be, if the Label control were not available. The Label control is a server-side control and can be manipulated using server-side code. Its text can be changed, along with formatting options and many other properties, which will be shown shortly.

Labels at Design-Time

Labels are very easy to use and manipulate, since they don't really contain much functionality. After all, all they do is display text. However, most of the properties of labels are available to the other, more advanced Web and Html Controls, so they're a good starting point and will introduce several properties that can be used on other controls. To show how the Html Control Label and Web Control Label are used, create a new Web Form and insert a Label Web Control and Label Html Control onto the form by dragging-and-dropping from the Toolbox. To make the Html Control Label run on the server-side, you need to right-click it and choose "Run As Server Control". Figure 5.5 shows the designer after the controls have been added and the Html Control converted to run as a server-side control.


WebForms - Microsoft Visual Basic .NET [design] - WebFormDemo.aspx
File Edit View Project Build Debug Data Format Table Insert Frames Tools Window Help
Debug
ViewState
Normal
Times New Roman
3
WebFormDemo.aspx
Label
Label
Toolbox
Solution Explorer
Properties
Design
HTML
Output
Ready

[Figure 5.5]

If the page were loaded in a browser now, the two Labels would look identical. However, when changing the properties of and manipulating the labels, the differences between the two are more apparent.

> At the moment, you might be wondering why the designer puts a grey border around the Html Control and not the Web Control Label. This is because by default the Html Control has a specific width in pixels set and the border shows the physical size of the control, whereas the Web Control Label is “free form” and will extend automatically according to the size of its contents. This is because of the different ways that the two types of label are rendered in HTML, and is a technical constraint of HTML.

Changing the properties of controls is very simple. As was shown in the previous chapter, the Properties Window in the IDE will display editable properties for the control that is currently selected. Selecting the Web Control Label will reveal numerous properties, including the following:

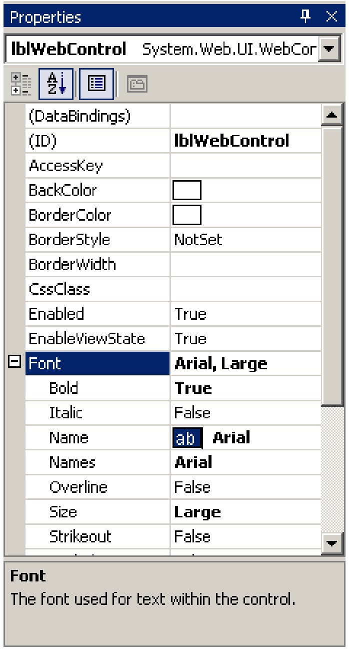
[Table 5.1]

Property	**Description**
ID	Sets the name/ID of the control
BackColor	Sets the background color of the control
BorderColor	Sets the border color of the control (if a border is enabled)
BorderStyle	Sets the style of a border for the control.
BorderWidth	Sets the width of a border for the control
CssClass	Sets the CSS class that will be applied to the control
EnableViewState	Sets whether the state of the control is maintained across postbacks or not
Font	A set of properties describing the font (including name, size etc)

ForeColor	Sets the color of the text
Text	Sets the actual text to be displayed
Visible	Sets whether the control is visible or not

These properties can be changed both at design-time (using the Properties Window, or by going into the HTML View and manually editing the properties) and at run-time (in server-side event handlers). The properties provide quite a lot of control over the appearance of the control, and some also influence its functionality. Most, if not all, of these properties are also available on the rest of the Web Controls.

Change the ID property to lblWebControl and modify the appearance of the control using the Properties window, as shown in Figure 5.6.



Properties

lblWebControl System.Web.UI.WebCor

(DataBindings)	
(ID)	**lblWebControl**
AccessKey	
BackColor	□
BorderColor	□
BorderStyle	NotSet
BorderWidth	
CssClass	
Enabled	True
EnableViewState	True
Font	**Arial, Large**
Bold	**True**
Italic	False
Name	ab **Arial**
Names	**Arial**
Overline	False
Size	**Large**
Strikeout	False

Font
The font used for text within the control.

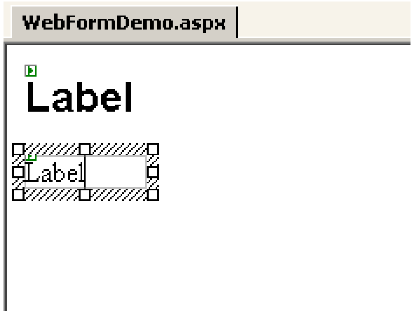
[Figure 5.6]

Notice that when the properties controlling visual aspects of the control are modified, the control is updated in the designer. Edit the Text property, so that “Hello World from a Web Control Label!” is displayed.

Select the Html Control Label, and you will notice that an almost entirely different set of properties is available, and a much smaller list at that. Table 5.2 lists the three most important ones.

Property	**Description**
ID	Sets the name/ID of the control
Class	Sets the CSS class that will be applied to the control
Style	Sets the inline CSS that will be applied to the control

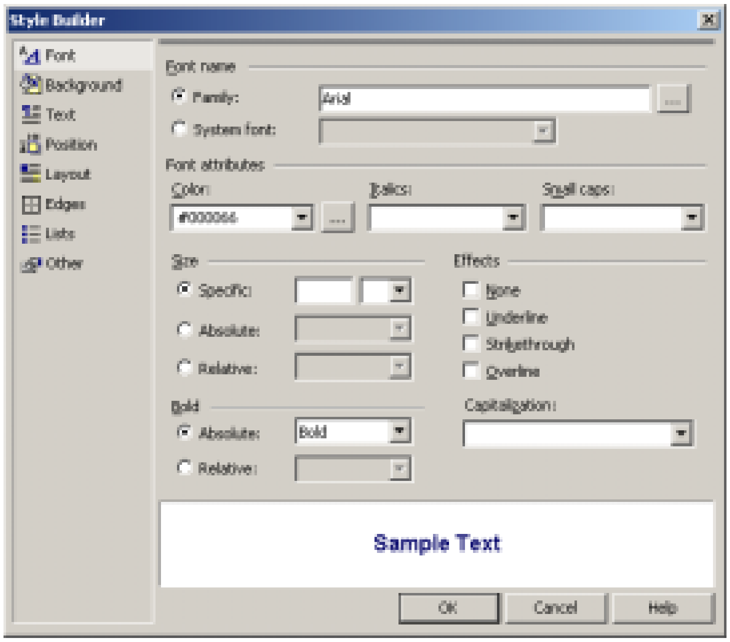
Besides the formatting properties, possibly the most notable omission is that of the “Text” property, which was used to set the text to be displayed in the Web Control Label. In the Html Control Label, the content of the control is actually stored “inside” the control (in the HTML markup, the text is shown between the opening and closing tags), and although a property is exposed for modifying this text at run time, at design time, the text is modified by selecting the control, and then clicking on the current text, as shown by figure 5.7:



[Figure 5.7]

Because the text is now being edited in the designer, rather than the Properties window, the Formatting Toolbar can be used to modify the appearance of the text (when a Web Control Label is selected, the Formatting Toolbar is disabled). The formatting is applied directly to the text using raw HTML (which is also included within the opening and closing tags of the Label), rather than using properties, as is the case with the Web Control Label. Formatting text is simple – select the text, and choose the appropriate formatting options, such as bold, italics, color or font name.

The control also has a “style” property, so it is possible to format the control using an inline style sheet. The Visual Studio designer provides a very useful “Style Builder” tool for this purpose, and is an alternative to using the formatting toolbar. To open the Style Builder, select the “style” property in the Properties window, and click on the “ellipsis button” on the right side of the item. Figure 5.8 shows the Style Builder after a few options have been changed to format the text as navy blue in color, Arial font and bold.



● Family: Arial

Color: #000066

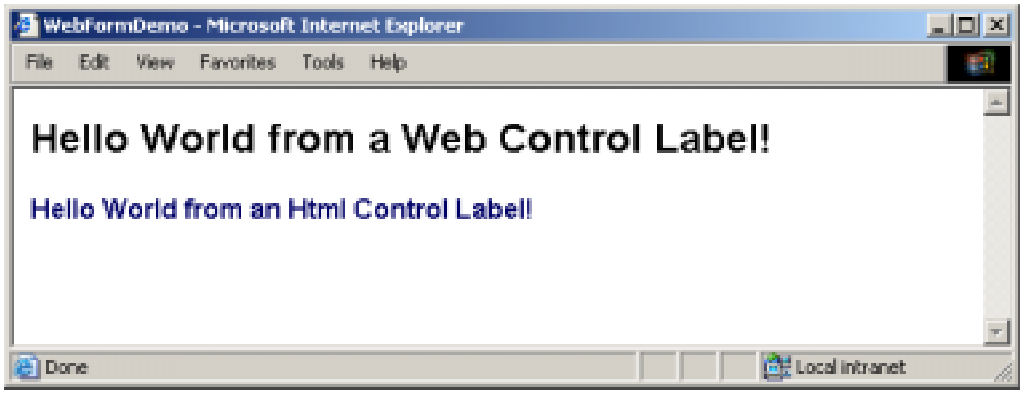
● Absolute: Bold

Sample Text

[Figure 5.8]

Using either the Formatting Toolbar, or the Style Builder, the same effects can be achieved with the Html Control Label as with the Web Control Label using its properties. For the purposes of this section, change the ID of the Html Control Label to “lblHtmlControl”, and modify its contents so that it displays the text “Hello World from an Html Control Label!” The control will probably need to be resized to fit the text on one line – the six “handles” surrounding the control when selected can be dragged using the mouse pointer to resize it.

After modifying the properties and styles of the two controls, figure 5.9 shows the page loaded in Internet Explorer.



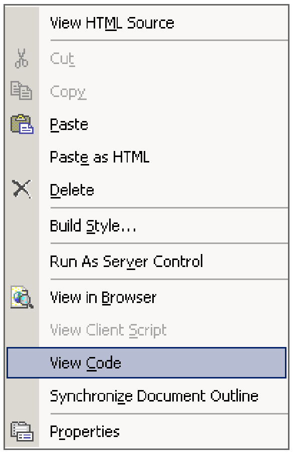
Hello World from a Web Control Label!

Hello World from an Html Control Label!

[Figure 5.9]

Changing the Labels at Run-time

As you already know, ASP.NET pages, or “Web Forms”, are split into two parts – the user interface, which is contained in a file with the extension .aspx, and the code-behind file containing all the server-side code for manipulating the objects in the UI, which is contained in a file with the extension .aspx.vb. There are several ways to switch from the UI designer to the server-side code, but the easiest is simply to right-click anywhere on the form and choose “View Code” from the context menu, as shown in Figure 5.10. You could also choose “View Code” from the context menu of the UI file in the Solution Explorer.



[Figure 5.10]

The “default” code that is displayed in the code-behind file (the code that is displayed for a blank Web Form) was explained in the previous chapter, but briefly, a class is created to represent the page, and each control on the form is declared as a variable so they can be accessed programmatically. One event handler is also automatically inserted – Page_Load. This is called each time the page is loaded, and is the perfect place to put in any code to initialize control values. The procedure looks like this:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  'Put user code to initialize the page here
End Sub
```

Although typically you'd set the text for a Label to display in the UI file rather than in the Page_Load event, it is certainly possible to do so. Listing 5.2 shows the Page_Load event handler that will change the text of the two Label controls (Web and Html) programmatically when the page loads.

[Listing 5.2]

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  lblWebControl.Text = "This is a Web Control Label"
  lblHtmlControl.InnerText = "This is an Html Control Label"
End Sub
```

The text of the Web Control Label will display "This is a Web Control Label", and the Html Control Label will display "This is an Html Control Label" when the page is viewed in the browser. However, it is important to take note that for the Web Control Label, the Text property was used, whereas the Html Control Label exposes the InnerText property, and doesn't have a Text property.

For both types of Label, all the properties listed in tables 5.1 and 5.2 can be programmatically modified by the respective control. Insert the following line into the Page_Load event handler, and the Web Control Label's background color will be silver when the page is viewed again:

```
lblWebControl.BackColor = Color.Silver
```

All "Color" properties (eg. BackColor, BorderColor, ForeColor etc) on all controls (not just labels) take a value of type Color. It is possible to generate custom colors, but generally you will choose a color from the selection in the System.Drawing.Color structure. These include Color.Black, Color.White, Color.Red, Color.Green, Color.Blue, Color.Gray and so forth. See the .NET Framework SDK documentation for a full list.

The Font property of the Web Control Label is however slightly different from the rest of the properties in that rather than accepting a single value, it actually contains a set of properties that describe the font. This is because the property is of type FontInfo, which is a class that is used to describe a font through its own properties. Listing 5.3 demonstrates the use of a couple of the Font properties.

[Listing 5.3]

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  lblWebControl.Text = "This is a Web Control Label"
  lblWebControl.Font.Name = "Courier New"
```

```
    lblWebControl.Font.Size = FontUnit.Point(16)
    lblHtmlControl.InnerText = "This is an Html Control Label"
  End Sub
```

This would change the font to “Courier New”, with a size of 16pts. The Font.Name property takes a string containing the name of the font that must be used, although the Font.Size property is slightly more complicated. It requires a FontUnit object be assigned to it. The System.Web.UI.WebControls.FontUnit structure contains a method, Point(), that should be passed an integer value representing the point-size of the font, and the method returns an instance of the FontUnit structure to represent this value.

Table 5.3 lists the FontInfo class’s properties that can be used to manipulate the Label’s font display using the Font property.

[Table 5.3]

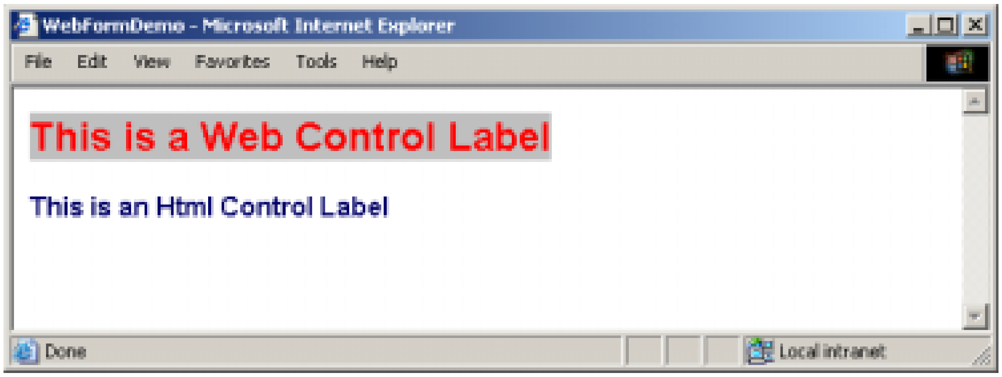
Property	**Type**	**Description**
Bold	Boolean	Sets whether the text should be bold or not.
Italic	Boolean	Sets whether the text should be italic or not.
Name	String	Sets the name of the font to use.
Names	String Array	Specifies an array of font names to use, in order of precedence.
Overline	Boolean	Sets whether the text should be overlined or not.
Size	FontUnit	Specifies the size of the text using a value from the FontUnit structure.
Strikeout	Boolean	Sets whether the text should be striked out or not.
Underline	Boolean	Sets whether the text should be underlined or not.

Both the Back- and Foreground colors of the Label can be changed using the Backcolor and Forecolor properties respectively. Both of these are of type System.Drawing.Color, which is a structure that defines a color. However, it also includes many pre-defined colors. Listing 5.4 shows how to programmatically change the back- and foreground colors of the Web Control Label by specifying the pre-defined colors red and silver.

[Listing 5.4]

```
  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
  System.EventArgs) Handles MyBase.Load
    lblWebControl.Text = "This is a Web Control Label"
    lblWebControl.BackColor = Color.Silver
    lblWebControl.ForeColor = Color.Red
    lblHtmlControl.InnerText = "This is an Html Control Label"
  End Sub
```

Figure 5.11 shows the result in Internet Explorer when the page is loaded.



This is a Web Control Label

This is an Html Control Label

[Figure 5.11]

Table 5.4 lists some of the more common colors that are included for use in the System.Drawing.Color structure.

[Table 5.4]

Color	Color	Color	
	Aqua	DarkViolet	Lime
	Beige	Fuchsia	Magenta
	Black	Gold	Maroon
	Blue	Gray	Navy
	Brown	Green	Orange
	Cyan	GreenYellow	Red
	DarkBlue	Indigo	Silver
	DarkGray	LightBlue	Transparent
	DarkOrange	LightGray	White
	DarkRed	LightYellow	Yellow

If a system-defined color doesn't exist for the color you require, there are several methods of "creating" your own custom color, but possibly the easiest is by using the Color structure's FromArgb() method. This method has several overloaded forms, but Listing 5.5 shows how it can be used to return a Color after having three integers passed in as parameters as the red, green and blue values for the color.

[Listing 5.5]

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  lblWebControl.Text = "This is a Web Control Label"
  lblWebControl.ForeColor = Color.FromArgb(245, 203, 10)
  lblHtmlControl.InnerText = "This is an Html Control Label"
End Sub
```

Colors can be created by mixing certain amounts of red, green and blue. In the RGB color model, the amount of the red, green and blue "color components" is specified on a scale of 0 through 255, where 0 is the least and 255 is the most. For example, to get a shade of purple, red and blue must be mixed, but no green, so an RGB value of 200, 0, 200 would define a purple color.

Using this method, any color can be created and used.

Programmatically modifying the appearance of Html Control Labels is entirely different from Web Control Labels, and is also significantly more restricted. We've already looked at the Html Control Label InnerText property, which allows you to modify the text that the label displays. However, because the control doesn't expose any "appearance properties", such as Font, ForeColor or BackColor, an alternative method must be used for changing these aspects of display. To this end, the Html Control Label provides the InnerHtml property. This allows the precise HTML code that will be placed in the output to the browser as the content of the control to be specified. It works very similarly to the InnerText property, excepting the fact that InnerText does not allow HTML tags to be entered as part of the Text, but InnerHtml does, so using this property the Label's text can be formatted directly using HTML formatting tags, such as FONT, I and B. Listing 5.6 shows how to programmatically change the text of the Html Control Label and make it italicized.

[Listing 5.6]

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  lblWebControl.Text = "This is a Web Control Label"
  lblHtmlControl.InnerHtml = "<i>This is an Html Control Label</i>"
End Sub
```

Although less abstracted than the Web Control Label method of formatting text, this still works. However, there is one particular major pitfall that will be encountered when using this method – if the formatting of the control is set via CSS (i.e. the Style Builder was used to format the control at Design Time), there is no way to programmatically modify those particular settings. The Label does expose a Style property at run-time, but this is read-only, so cannot be changed. The issue lies in the precedence that HTML gives to CSS over regular HTML formatting tags – when CSS is used, HTML will always use the formatting provided in the style over any formatting using HTML tags. If however the style doesn't specify a value for a particular type of formatting, the HTML formatting tag will be applied. For example, if the style specifies that the color of the text must be red, and that it must be bold, there is no way that the color can be changed or the text can have the bold removed, but it will still be possible to make the text underlined and change the size of the font.

Needless to say, this is a very significant weakness in the Html Control Label, and as such it would probably be better to simply using the Web Control Label when a Label is needed.

Deleting the Controls

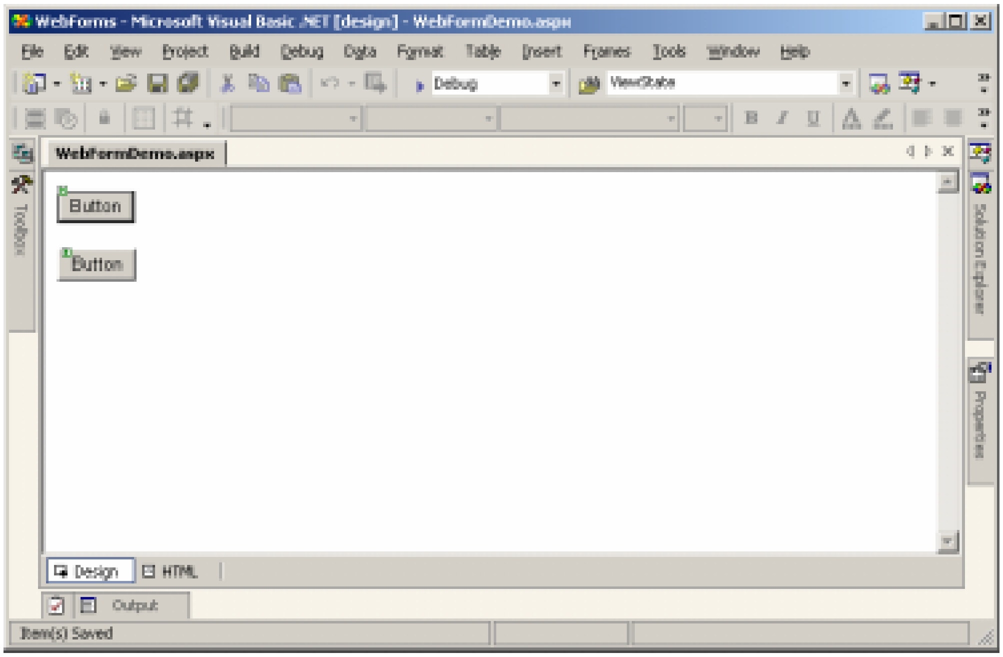
If you need to remove a Label from the form (or any other type of control), you simply right-click on it in the Web Form designer, and click "Delete" on the context menu that appears. Take note though, that even if you remove a control from the Web Form, any references to it in the server-side code will not automatically be removed, so if you remove a control that is programmatically modified in the Page_Load event handler, you will need to manually remove the lines in that event handler that involve that specific control, otherwise the application will return an error saying that it can’t find the control you’re attempting to reference when you next try to run the application. Incidentally, the same applies to when you rename Label controls (by changing their ID properties in the Properties Window), as any references to the old control names in the code-behind file will not be changed automatically, and you will need to manually modify the code to ensure that the new names are used.

The Button Controls

Simply put, the entire purpose of a Button Server Control is to cause the page to be reloaded in the browser, which is known as a ‘postback’ in ASP.NET However, the Button controls can also have server-side event handlers that fire when the reload occurs, and can therefore operate very similarly to the way in which they do in normal Windows application development.

A critical concept in ASP.NET development, most evident when using the Button control, is that when the form is submitted in the browser (normally through a Button Control being clicked), the data in the form is sent back to the same file on the web server as it was sent from. This process is called a “postback”, because the data in the form is “posted back” to the original file. In traditional ASP this technique was also used, but the ASP model didn’t specifically encourage it so much of the ‘plumbing’ had to be redone for every page. Because of this, the ‘postback’ technique wasn’t used a lot of the time, and pages would submit data to a different file. However, ASP.NET attempts to achieve a more Windows application-like programming experience by making each page a self-contained unit, just as a form in a Windows application.

Figure 5.12 shows a page in design mode with two Button controls added – a Web Control on the top, and an Html Control underneath. Since the Html Control Button is not automatically a server-side control, it has also been right-clicked and its “Run As Server Control” option checked. Create a new page, so that it looks like this.


WebForms - Microsoft Visual Basic .NET [design] - WebFormDemo.aspx
File Edit View Project Build Debug Data Format Table Insert Frames Tools Window Help
Debug
ViewState
WebFormDemo.aspx
Toolbox
Button
Button
Solution Explorer
Properties
Design
HTML
Output
Item(s) Saved

[Figure 5.12]

At first there appears to be no difference between the two controls, but again a look at the Properties Window for each shows otherwise. The appearance of the Web Control Button can be controlled using the same properties as for the Label Web Control (such as ForeColor, BackColor etc), whilst the Html Control Button must again use inline styles using the “style” property. However, possibly the most important difference between the two property sets for the moment is the property that controls the caption of the button – for the Web Control Button it is “Text”, but for the Html Control Button, it is “value”. Modify these two properties to “Web Control Button” and “Html Control Button” respectively, and then change the respective ID properties to “btnWebControl” and “btnHtmlControl”.

Setting up Event Handlers

ASP.NET revolves around an event-driven programming model, so one of the most important aspects of ASP.NET programming is the setting-up of event handlers to handle events fired by controls on your Web Forms. The Button control’s most commonly used event is “Click”, which occurs when the button is clicked. Visual Studio.NET can be used to quickly and easily set up an event handler for this event by simply double-clicking the button that you want an event handler to be set up for. Visual Studio.NET will then create the blank subroutine that will handle the event and will switch to code view. Double click on the Web Control Button to demonstrate this. Listing 5.7 displays the blank event handler that will have been created.

[Listing 5.7]

```
Private Sub btnWebControl_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnWebControl.Click


End Sub
```

What has in effect been created is a subroutine, btnWebControl_Click, that is called every time the btnWebControl Button is clicked. Visual Studio.NET’s convention for event handler names is the control name (in this case btnWebControl), followed by an underscore (_) and finally by the name of the event that the subroutine will be handling. Different types of events may pass different types of information to subroutines that handle them as parameters, but the Button Click event is fairly simple and only passes the object that caused the event to fire (sender) and an instance of the EventArgs class (e). The use for these two parameters will become evident later, and they can actually prove to be very useful in certain circumstances.

However, the part of the event handler declaration that differentiates it from any other normal subroutine declaration is the presence of the Handles keyword. This keyword marks the subroutine as the event handler for the list of events that follows it. In this case, the event handler only handles one event, btnWebControl.Click.

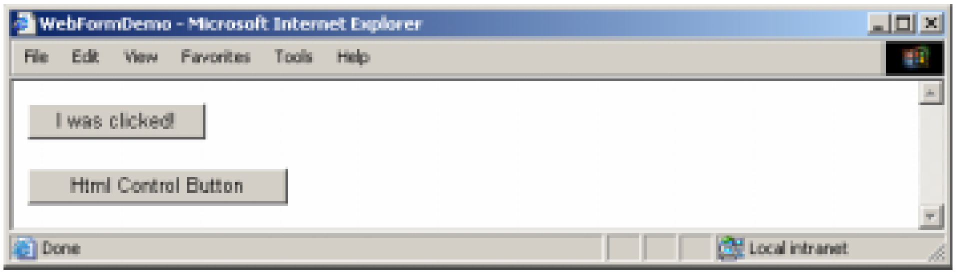
Put the following line of code into the event handler subroutine:

```
btnWebControl.Text = "I was clicked!"
```

This will make the button’s caption change to “I was clicked!” when the user clicks on it. Create an event handler for the Html Control Button, also by double-clicking on it in design view, and then place the following line into it:

```
btnHtmlControl.Value = "I was also clicked!"
```

Similarly this will make the Html Control Button's caption change to "I was also clicked!" when it is clicked. Load the page in the browser and click on the first button. Figure 5.13 shows what should be displayed.



[Figure 5.13]

This is probably the result that both ASP and VB programmers would have expected, but clicking the "Html Control Button" may just result in a very pleasant surprise for the previous ASP programmers. Give it a try. Nothing momentous happened, as the Html Control Button's caption changed as expected, and the VB programmers might be wondering what the pleasant surprise is. When the 2nd button was clicked and its caption changed, the 1st button's caption didn't revert back to "Web Control Button", but rather stayed as "I was clicked!" Previously in ASP it would have been necessary to manually re-assign that value to the button, but ASP.NET automatically maintains the state of controls across postbacks, so there's no longer any need to re-assign values to controls once they've already been assigned.

Where did my Submit Button go?

The ASP programmers amongst you might be wondering where the submit button is on the form, and how the form submits without a submit button. In fact, the answer in the case of the Button Web Control is quite simple. As you know, all ASP.NET Server Controls output regular HTML to the browser when they're rendered - not specialized markup, or anything fancy, just plain HTML. The Button Web Control actually renders as an HTML submit button, using the <input type=submit> tag. If you view the source of the page in Internet Explorer, you'll be able to verify this.

However, the Button Html Control is slightly more complicated – it renders as an HTML button using the <input type=button> tag, which you, if you're an HTML guru, will know doesn't submit the form automatically. In this case, ASP.NET creates a client-side JavaScript that submits the form when the button is clicked.

So, whether you're using the Button Html or Web Control, the form will be submitted in a similar fashion to if a regular submit button was used.

The Page Cycle

Understanding the page cycle is very important and will help you develop more robust ASP.NET applications. This section will give you a very brief overview of what's actually happening when a button is clicked and a postback occurs.

When the page is first loaded in the browser, the following steps occur:

1. Browser sends request for page
1. 2. Server receives request, ASP.NET loads the page, and the Page_Load event is fired
3. The Page_Unload event is fired and the HTML response is sent back to the browser.

However, the process for postbacks is slightly more complicated, as now the controls' values must be restored from what they were last set to, and events must be raised (eg. If a button was clicked to cause the postback, the button's Click event must be raised, and then the handler executed if it exists):

1. Browser sends request for page
1. 2. Server receives request and ASP.NET loads the page
2. 3. All the values for the server controls are restored to what they were
3. 4. The Page_Load event is fired
4. 5. ASP.NET fires all the event handlers for the controls (such as a Button's Click event). It is inside the event handlers that the values that the user has entered into the form are used.
5. 6. The values of all the server controls are saved, as they may have been changed in the code inside event handlers.
7. The HTML response is sent back to the browser and the Page_Unload event is fired.

Therefore by the time the Page_Load event is fired, all the controls will have had their "state" restored, and this all happens automatically. After the Page_Load event has fired, all the other events relating to the controls on the form are fired (such as a Button's Click event).

ASP.NET stores the values of the controls using a technique known as "ViewState". It does so by placing a hidden variable in the form in the HTML that is returned to the browser containing the values that must be restored on the next postback. For example, load your page with the two buttons, click one of them, and then click View-Source in Internet Explorer. A Notepad window should appear showing the HTML that ASP.NET generated. One of the lines should look something like this:

```
<input type="hidden" name="__VIEWSTATE"
value="dDwtODg0MTA4NDE3O3Q8O2w8aTwx
Pjs+O2w8dDw7bDxpPDE+Oz47bDx0PHA8cDxsPFRleHQ7PjtsPEkgd2
FzIGNsaWNrZWQhOz4+Oz47
Oz47Pj47Pj47Pg==" />
```

This is an encoded string containing the values of the variables that ASP.NET must restore on the next postback. When you click the other button and ASP.NET loads the page, it receives this string in the request, decodes it and assigns the value to the 1st button so that it's value is "remembered", even though the page didn't explicitly restore it – ASP.NET handled it in the background. In the next chapter we'll advance on this and show how to put your own custom information into ViewState.

Page.IsPostBack

Quite often it’s very useful to be able to perform operations only the first time the page is loaded. Normally these will occur in the Page_Load event, and will consist of setting control properties or other variable values. However, code in the Page_Load event will ordinarily be executed every time the page is loaded, not just the first time, which could obviously have very strange effects on the functioning of the application as every time a button is clicked, or a postback occurs otherwise, the default values will be reset. However, the Page object includes a property, “IsPostBack” that returns a Boolean value – true is the page was loaded through a postback, and false if not. Since a postback occurs (normally through a button being clicked) when a page sends a request back to the server for itself in order to be updated, the only request that is not a postback is the initial, first request for the page. So it is therefore possible to ascertain whether the page is being loaded for the first time by using the Page.IsPostBack property. Listing 5.8 shows a sample Page_Load event handler where the code will only be executed the first time the page is shown. In this code, a button, btnWebControl, has its caption set.

Listing 5.8

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  If Not Page.IsPostBack Then
    btnWebControl.Text = "Default Value"
  End If
End Sub
```

Take note that the If condition uses the Not operator, as the page must not be a postback. Since if it was, this would mean that the page isn’t being loaded for the first time. This technique is a very useful tool, and although it may not seem so now since you can easily set control properties in the IDE, and there is no need to set them programmatically, it is often necessary to do so when inserting values from a database, for example, as these obviously cannot be set at design time.

The TextBox Controls

The Html and Web TextBox Controls are the foremost controls for gathering user input. Their sole purpose is to allow the user to enter data into the application, and where a generic user input control is required, the TextBox will normally be perfectly suited to the task. In essence, the TextBox allows the user to enter a text string, and when a postback occurs, this string can then be processed. For input where there are no choices (such as yes/no, or a specific list of possible options) and the input from each user will often be entirely different, such as an input of the user’s name, a TextBox will be used.

As with the previous controls, the basic functionality of the Html and Web varieties of the control are almost identical, but there are numerous properties that the Web Control supports that the Html Control doesn’t. For example, as with the Label controls, the TextBox WebControl includes a Font property for modifying various aspects of how the font it uses, and also includes ForeColor and BackColor properties for defining the foreground and background colors that the control should use.

However, again as with the Label control, the Html Control TextBox only provides the style property for modifying its appearance, which can be configured using the Style Builder.

To demonstrate the use of the TextBox controls, place one Web Control TextBox and one Html Control TextBox (the control is labeled as a “Text Field” in the Toolbox) onto a blank Web Form, underneath each other. Remember, to make the Html Control a proper server-side control, you need to right-click it and click “Run As Server Control”. Place a Web Control Label and a Web Control Button beneath these controls as well. Figure 5.14 shows what the form should now look like in the designer.

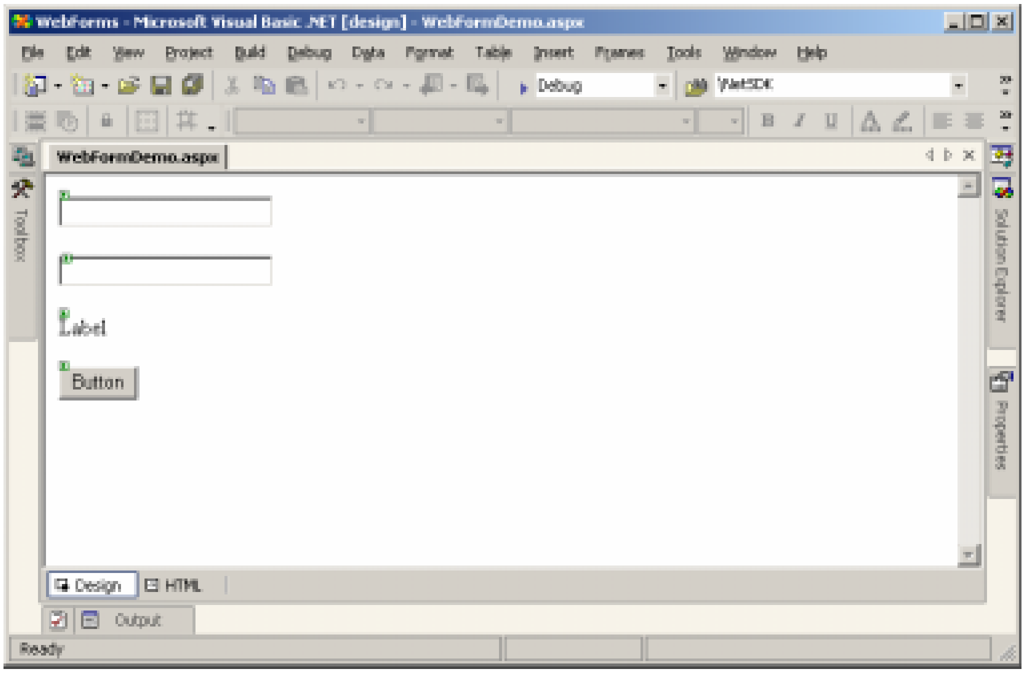
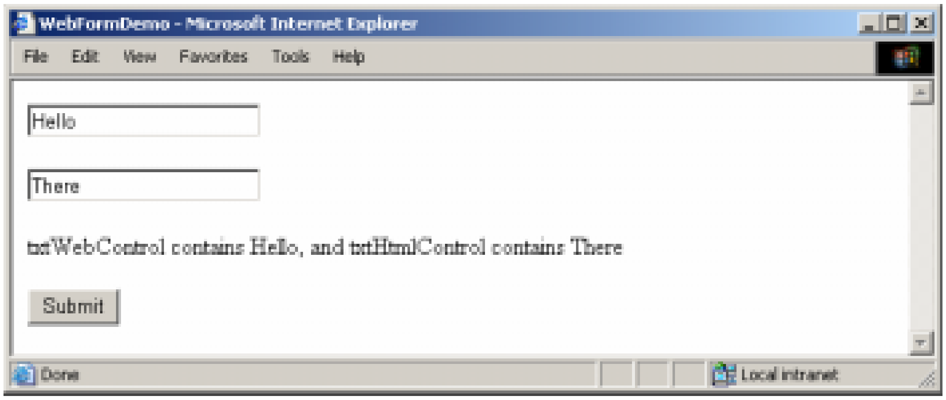


Figure 5.14 – The Web Forms Designer with several controls added

Using the Properties Window, set the ID property of the controls to txtWebControl, txtHtmlControl, lblDisplay and btnSubmit respectively. Change the Text property of lblDisplay to nothing, and of btnSubmit to “Submit”. Double click on the Button so that a Click event handler is hooked up, and the server-side code is displayed. We’ll now insert code so that when the button is clicked, the label will display the contents of both TextBox controls. Insert the following line of code into the event handler:

```
lblDisplay.Text = "txtWebControl contains " & txtWebControl.Text & ", and "

_

& txtHtmlControl contains " & txtHtmlControl.Value
```

As with the Label controls, accessing the contents of the TextBox controls is slightly different for the Html and Web Control varieties – the Web Control exposes a Text property for this purpose, and the Html Control has a Value property. However, in this example we’re extending the functionality slightly by not simply assigning the text of the label to be the value of a TextBox, but are concatenating, or joining, the values from both TextBox controls together, along with a portion of static text. The ampersand (&) is the concatenation operator in Visual Basic.NET and can be used to join various strings together. Figure 5.15 shows the page loaded after values have been entered into the two TextBoxes and the button has been clicked.



Hello

There

txtWebControl contains Hello, and txtHtmlControl contains There

Figure 5.15 – Displaying the contents of two TextBox controls in a Label

The StringBuilder Class

In the above example, we join several strings using the ampersand. However, in VB.NET this is an extremely inefficient way of joining strings. If you're building a small application that isn't going to be used by many people, then this performance hit isn't going to be noticeable at all. However, if the application is expecting many users, or the string joining procedure is occurring inside a loop, then you should rather use the StringBuilder class to join the strings, as I'm about to show you.

We'll continue with the example of joining a few static strings and the contents of two TextBox controls, so delete the line currently in the btnSubmit click event handler, and we'll rewrite the code to perform the string joins using a StringBuilder object. Listing 5.9 shows the code, and we'll dissect it line by line.

Listing 5.9

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
  Dim objStringBuilder As New System.Text.StringBuilder()
  objStringBuilder.Append("txtWebControl contains ")
  objStringBuilder.Append(txtWebControl.Text)
  objStringBuilder.Append(" and txtHtmlControl contains ")
  objStringBuilder.Append(txtHtmlControl.Value)
  lblDisplay.Text = objStringBuilder.ToString()
End Sub
```

In the first line, we are creating a new StringBuilder object, and a reference to it called objStringBuilder. As you know, all classes in .NET are organized into namespaces, and the StringBuilder class is in the System.Text namespace. Now that we have an instance of the StringBuilder class, we can work with it. The StringBuilder's Append method has many overloaded versions (i.e. it can accept many different types and numbers of parameters), and the one we are using accepts only one string as a parameter. As the name suggests, the StringBuilder is used to build strings, and when the Append method is called, it adds the string in the parameter to the string that is being created (but without the performance hit of using the ampersand to concatenate the strings).

Since our string join is fairly simple, we only need to use the Append method. However, Insert and Remove methods are also available to, as their names suggest, insert substrings into the string that is being built at specific positions, and remove parts of the string being built.

Once the string has been built, calling the StringBuilder's ToString method returns the string. In this case, the string is assigned to the lblDisplay Label's Text property. When the page is viewed in IE and the same data is inputted, exactly the same result will appear as when the string was created using the ampersands. It's easy to be lazy and avoid using the StringBuilder for string joining since it requires more effort than the operator method, but remember that the performance of the StringBuilder over concatenation using the ampersand operator is exponentially greater when many joins are being performed, so for real applications, try to avoid using the ampersand for joins wherever possible.

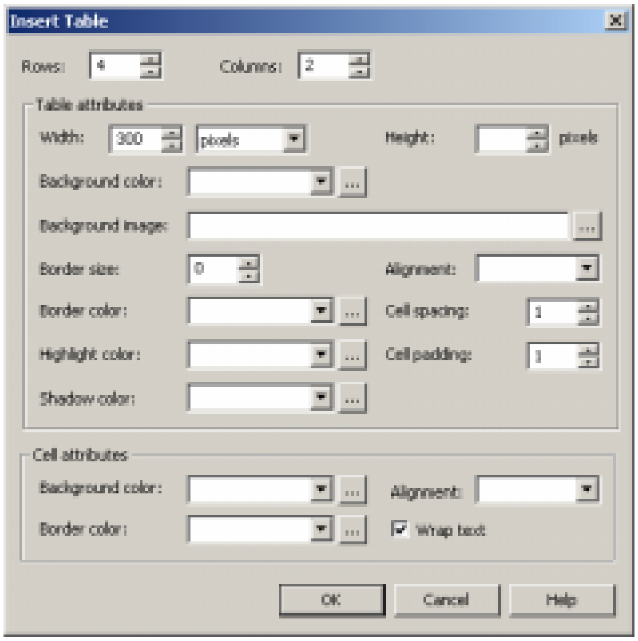
Input Layout using Tables

When you're getting input from a user, be it with a TextBox, or any one of the other input controls we'll discuss in this chapter, you'll typically include a caption next to the control so that the user knows what data or information he/she should provide. To make the layout more aesthetically pleasing, it's good practice to use a 2-column table – the 1st column for the captions, and the 2nd column for the input controls. Fortunately Visual Studio.NET provides functionality in the designer that makes designing tables simple.

To practice, we're going to take the page we've created in the first part of this section and modify it so that all the controls appear within a table, and each TextBox control has a caption.

Firstly, we need to add a table to the page. To do so, move the cursor in the designer to the top left of the page (the easiest way to do this is by pressing Ctrl-Home on your keyboard). In the main menu, click "Table", then "Insert", then "Table". A dialog should appear allowing you to configure the table that will be inserted. Change the "Rows" to 4, the "Columns" to 2, and the "Border size" to 0. Figure 5.16 shows what the dialog should look like after changing those settings.

Figure 5.16 – Insert Table Dialog



Insert Table

Rows: 4 Columns: 2

Table attributes

Width: 300 pixels Height: pixels

Background color:

Background image:

Border size: 0 Alignment:

Border color: Cell spacing: 1

Highlight color: Cell padding: 1

Shadow color:

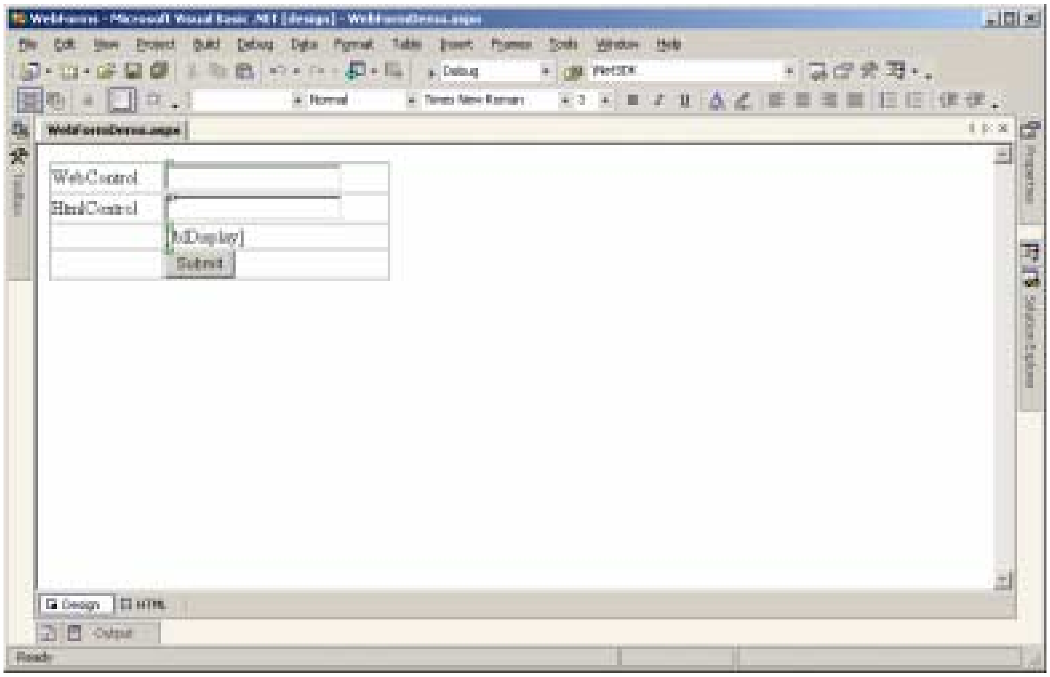
Cell attributes

Background color: Alignment:

Border color: ☑ Wrap text

Click “OK”, and the table should be inserted at the top of the page. Click in the first cell (top left), and type in “WebControl”. Click in the left cell in the second row and type “HtmlControl”. You will notice that the second column has suddenly shrunk – this is simply because of the way in which tables are rendered in HTML, and when content is inserted the column with “grow” again. Drag and drop the first TextBox, txtWebControl, into the second cell in the first row. You will notice that the column has resized to accommodate the control. Drag and drop the rest of the controls into the second table column, each in their own cell, one underneath each other. Figure 5.17 shows how the page should look in the designer.

Figure 5.17 – Layout out an input form using a table



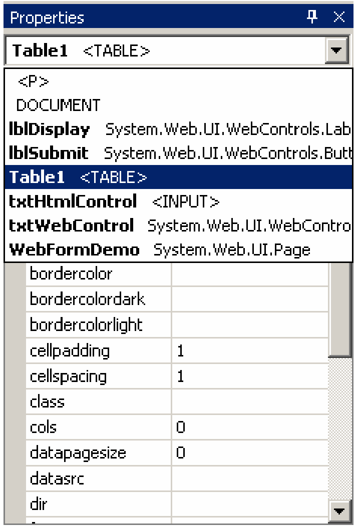
WebForms - Microsoft Visual Basic .NET [design] - WebFormsDemo.aspx

WebFormsDemo.aspx

WebControl	
HtmlControl	
	[lblDisplay]
	Submit

By default the table has a set width of 300 pixels, which is useful when designing the form since the cells are a reasonable size and are easy to drop controls into or move the cursor into to type captions (take note that you simply type in the text in the designer for captions, and don't use Label controls). However, once you've finished designing, it is better to simply remove the table's default width and let the table automatically adjust its size according to its contents. To do this, select the table from the Properties dropdown in the Properties dialog, as shown in Figure 5.18, and clear the text in the "width" property.

Figure 5.18 – Selecting a specific control in the Properties Window



Properties

Table1 <TABLE>

<P>
DOCUMENT
lblDisplay System.Web.UI.WebControls.Lab
lblSubmit System.Web.UI.WebControls.Butt
Table1 <TABLE>
txtHtmlControl <INPUT>
txtWebControl System.Web.UI.WebContro
WebFormDemo System.Web.UI.Page

bordercolor	
bordercolordark	
bordercolorlight	
cellpadding	1
cellspacing	1
class	
cols	0
datapagesize	0
datasrc	
dir	

You can now run the application, which should display the same message as before when the Submit button is clicked, but now the fields are nicely in line, and have captions, labelling each one.

The Table Web and Html Controls are covered later in the chapter, which allow you to programmatically modify the table, but for layout purposes the plain, non-server control table is adequate.

Deleting a Table

If you need to delete a table in the designer, the easiest way to do it is by placing the cursor to the immediate right of the table and pressing, "Backspace" on your keyboard. Remember that if you delete a table, you automatically delete all the controls contained in the table, but any references to the controls inside the server-side code will not automatically be removed, and needs to be done manually.

Performing Data Type Conversions on TextBox Inputs

TextBox controls can be used to gather any type of data – it might simply be a string, but it could be an Integer, or a floating-point number, or a date. Since the Text and Value properties of the TextBox controls automatically assume that the contents of the control is a string, in order to be able to work with and manipulate inputs that aren't strings, they must first be converted to their correct data type. Likewise, when results are displayed in Label controls (or in any other control), a string is normally expected, so another conversion must take place to convert the result from its data type to a string before it can be displayed.

Using our existing form containing two TextBox controls, one a Web Control and another an Html Control, a Label and a Button, we'll build a tool that adds two Integers, entered in the TextBox controls, together, and displays the result in the Label. It's a very simple task, but it requires that three type conversions take place – both inputted numbers must first be converted to type Integer, and then once the result of the addition has been calculated, that must be converted to a string before it can be displayed using the Label.

In Visual Basic.NET there are two ways to perform type conversions (and in some cases, three). The first is specific to the VB.NET language and won't work in any of the other .NET languages, and the second is through using a class provided for this specific purpose in the .NET Base Class Library, which is available to all .NET languages.

Visual Basic and VBScript developers alike will recognise the intrinsic VB functions such as CInt, CStr and so on. These are again available in Visual Basic.NET, and work in exactly the same way as before, so you should feel right at home with them. For those of you who haven't encountered these conversion functions before, they all accept one parameter of any basic type (such as string, Integer, Long, Date etc), and will return that variable cast as the type that their namesake suggests. For example, the CInt function could have a string passed into it, and it will convert that string to an Integer and return that value. Likewise, CStr could have an Integer passed into it, and the string representation of the Integer will be returned. Table 5.5 shows the full list of these functions.

Table 5.5 – Intrinsic VB.NET Type Conversion Functions

Function	**Converts Parameter To**
CBool	Boolean
CByte	Byte
CChar	Char
CDate	Date
CDbl	Double (double-precision floating-point number)
CDec	Decimal (floating-point number)
CInt	Integer
CLng	Long (long Integer)
CObj	Object
CShort	Short (short Integer)
CSingle	Single (single-precision floating-point number)
CStr	String

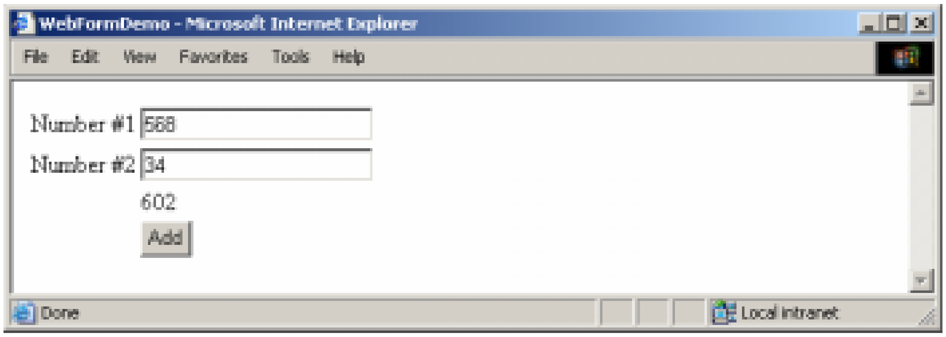
In the Web Form designer, change the name/ID of txtWebControl to txtNumber1, and txtHtmlControl to txtNumber2. Change the Text property of btnSubmit to "Add". If you've inserted captions for the TextBox controls, you can change them to "Number #1" and "Number #2" respectively. Open the server-side Click event handler for the btnSubmit Button and remove any code inside the subroutine, and replace it with Listing 5.10.

Listing 5.10

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
  Dim intNumber1, intNumber2, intAnswer As Integer
  Dim strAnswer As String
  intNumber1 = CInt(txtNumber1.Text)
  intNumber2 = CInt(txtNumber2.Value)
  intAnswer = intNumber1 + intNumber2
  strAnswer = CStr(intAnswer)
  lblDisplay.Text = strAnswer
End Sub
```

Try loading the page and entering in two numbers to add and clicking "Add". Figure 5.19 shows a sample of how the page should look.

Figure 5.19 – Adding two Integers and displaying the result



Number #1 568

Number #2 34

602

Your first reaction may be, "Why are there 7 lines of code just to add two numbers together?" However, the code could easily be reduced to a one-liner, but it's in an extremely verbose form here to separate each conversion that needs to take place. First, three variables of type Integer are declared – they will store the two numbers to be added and the answer. Second, a string variable is declared, which is where the answer will be stored in string form. Next, the first number is converted from a string in txtNumber1.Text to an integer using the CInt function, and the result is assigned to the variable intNumber1. Similarly the number in txtNumber2.Value (to show that these functions work with strings from any control) is cast as an Integer and assigned to intNumber2. intAnswer is then assigned as the sum of intNumber1 and intNumber2. The next line uses the CStr function to convert the intAnswer Integer to a string, which is assigned to the strAnswer variable, and this is assigned to lblDisplay's Text property.

Obviously all these variables aren't required, since they're all only used once each and don't need to be manipulated multiple times, so this whole code block can be condensed into the following line of code:

```
lblDisplay.Text = CStr(CInt(txtNumber1.Text) + CInt(txtNumber2.Value))
```

Using the .NET Framework conversion functions is equally easy. They are all stored as static, or shared, methods of the System.Convert class, and operate very similarly to their intrinsic VB.NET counterparts. Table 5.6 lists the most commonly used methods in Convert class.

Table 5.6 – Commonly used System.Convert Type Conversion Methods

Method	**Converts Parameter To**
ToBoolean	Boolean
ToByte	Byte
ToChar	Char
ToDateTime	Date
ToDouble	Double (double-precision floating-point number)
ToDecimal	Decimal (floating-point number)
ToInt16	16-bit Integer
ToInt32	32-bit Integer
ToInt64	64-bit Integer
ToSingle	Single (single-precision floating-point number)
ToString	String

Listing 5.11 shows the same code as Listing 5.10, except all the conversions are now done using the Convert class.

Listing 5.11

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
  Dim intNumber1, intNumber2, intAnswer As Integer
  Dim strAnswer As String
  intNumber1 = Convert.ToInt32(txtNumber1.Text)
  intNumber2 = Convert.ToInt32(txtNumber2.Value)
  intAnswer = intNumber1 + intNumber2
  strAnswer = Convert.ToString(intAnswer)
  lblDisplay.Text = strAnswer
End Sub
```

As you can see, the numbers from txtNumber1.Text and txtNumber2.Value are converted into Integers using the Convert.ToInt32 method (VB.NET Integer variables are 32-bit Integers), and the Convert.ToString method is used to convert the answer back to a string. Again, this can also be done as a one-liner, as follows:

```
lblDisplay.Text = Convert.ToString(Convert.ToInt32(txtNumber1.Text) +
Convert.ToInt32(txtNumber2.Value))
```

Data type conversions are very important in .NET programming, since VB.NET and the other .NET languages ensure type safety by requiring that for the most part type casting should be done explicitly, either using the VB.NET-specific functions or using the Convert class' methods.

There are two pitfalls that you should watch out for when typecasting though. Firstly, you may encounter problems, particularly with the number data types, when the type you're converting can hold a larger value than the type you're converting to.

For example, if you convert a 32-bit Integer to a Byte (8-bit Integer), or a 16-bit Integer, it is very possible that you'll receive an overflow error because the number you're trying to cast into a 16-bit or 8-bit Integer is larger than the capacity of that particular data type. The second problem you may encounter is that some data types have certain restrictions about what they can hold that others don't, and this may also cause errors to be raised. For example, a string containing "Abc" obviously can't be converted into an Integer. Another important point to remember is that when converting from a floating-point number type to an Integer type, the decimal point obviously can't be kept and is rounded. Chapter 7, "User Input Validation" shows you how to ensure that users enter data correctly, so if an Integer is expected, the user cannot enter anything except for an Integer, for example.

The Multi-line Text Input

The multi-line text input is used when the user should be able to enter a large amount of text, including new lines. This is most often seen in "Comments" fields, where the user will often enter a paragraph or two of text, and a regular single-line TextBox would not be appropriate.

The Web Controls and Html Controls collections diverge somewhat on there implementation of the multi-line text input – if you want to use a Web Control, then you use the TextBox control and set a property to make it multi-line, but there is an entirely separate Html Control, the Text Area control, for this purpose. However, despite this difference, their basic functionality is identical.

Starting with a blank Web Form, add a Web Control TextBox. Set it's ID property to "txtWebControl", and it's TextMode property to "MultiLine". You'll notice that the control has expanded, and now had two rows. To modify the number of rows and columns that are available in the input area, use the Columns and Rows properties. "Columns" is the number of characters that can be entered in one row, since by default in HTML the multi-line input uses a fixed-width font.

Add a "Text Area" control from the Html tools palette beneath txtWebControl, and set it's ID property to txtHtmlControl. Its "cols" and "rows" properties are used to set the number of columns and rows in the input area respectively. Add a Button Web Control, assign its ID as "btnSwap" and change its Text property to "Swap Text".

The multi-line text controls work almost identically to the TextBox controls, so the Web Control obviously still uses the Text property to assign or retrieve the text in the control, and the Html Text Area Control uses the Value property for this purpose. Double click on btnSwap to create a Click event handler for it. Listing 5.12 shows the code to swap the text in the two inputs around, demonstrating that indeed manipulating the content of the multi-line text controls is the same as for the single-line controls.

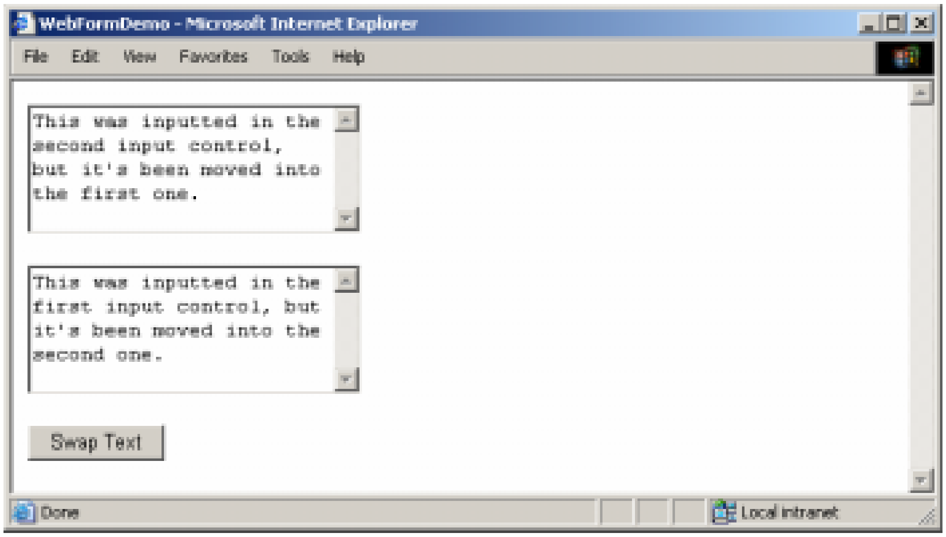
> If when you type out the code below, and all references of the txtHtmlControl object are underlined in blue, it probably means that you've forgotten to right-click the TextBox Html Control in the designer and choose "Run As Server Control". When a statement is underlined in blue in the code editor, it means that VS.NET doesn't recognize the statement because it hasn't yet been defined. In this case, since the TextBox Html Control hasn't been set to be available for manipulation on the server side, a declaration for it has not been added to the top of the Page class, and therefore VS.NET can't find a reference to it.

Listing 5.12

```
Private Sub btnSwap_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSwap.Click
  Dim strTemp As String
  strTemp = txtWebControl.Text
  txtWebControl.Text = txtHtmlControl.Value
  txtHtmlControl.Value = strTemp
End Sub
```

Figure 5.20 shows the two multi-line inputs, after text has been entered and then swapped.

Figure 5.20 – The Multi-line Text Input Controls



WebFormDemo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Swap Text
Done
Local intranet

This was inputted in the second input control, but it's been moved into the first one.

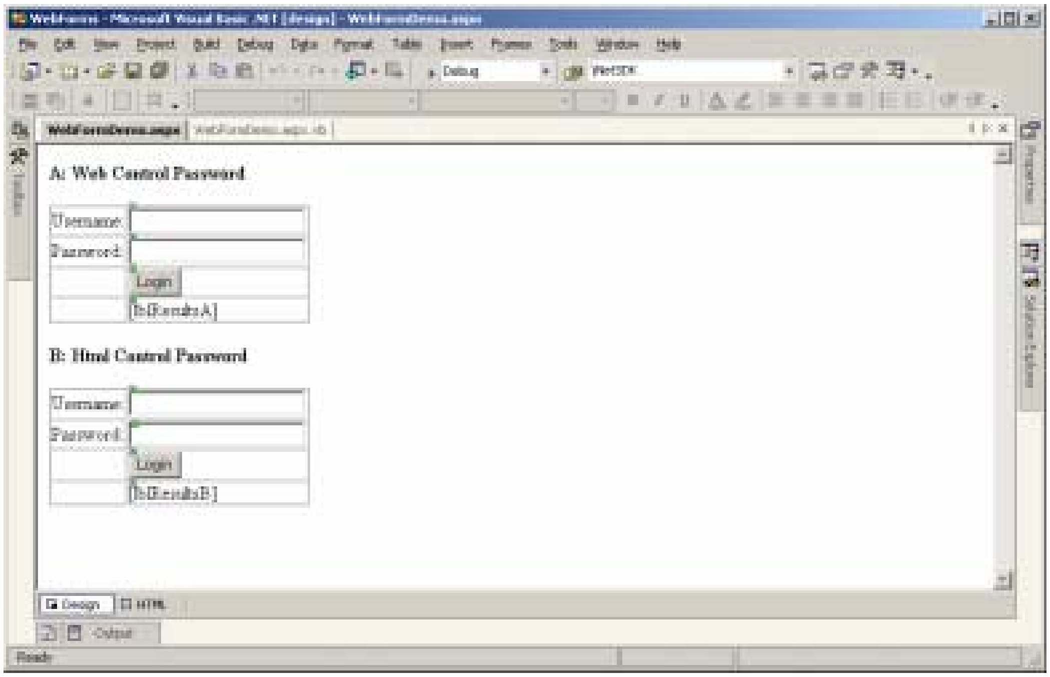
This was inputted in the first input control, but it's been moved into the second one.

The Password Field

The password input field is ubiquitous in the modern-day web, and provides protection against one of the main ways that passwords are stolen – passers-by simply glancing at the screen whilst a user is entering a password. The password field is simply a TextBox that stores the text that is being entered, but doesn't display it as it is being typed – asterixes (*) are used in place of the text. However, obviously when the form is submitted, the actual text that was entered is sent to the server, not to string of asterisks.

To demonstrate the password field, we'll use it in its most common form – as part of a login. As with the multi-line input, the password field is implemented as a TextBox Web Control, but has its TextMode property changed to "Password". However, there is an entirely separate "Password Field" Html Control. Both perform the same task, and are for all intents and purposes essentially identical. Figure 5.21 shows a web form in the VS.NET designer where in part A, a TextBox Web Control is used as the password field, and its TextMode property is set to "Password", but in part B, the password field is an Html Control.

Figure 5.21 – A login form in the designer


WebForms - Microsoft Visual Basic .NET [design] - WebFormsDemo.aspx
File Edit View Project Build Debug Data Format Table Insert Frames Tools Window Help
Debug
WebSDK
WebFormsDemo.aspx
WebFormsDemo.aspx.vb
Toolbox
Properties
Solution Explorer
Design
HTML
Output
Ready

A: Web Control Password

Username:
Password:
Login
[lblResultsA]

B: Html Control Password

Username:
Password:
Login
[lblResultsB]

In part A, the controls’ IDs are txtUsernameA, txtPasswordA, btnLoginA and lblResultsA, and similarly in part B, but with B replacing the trailing A in each case. Listing 5.13 shows the code for the Click event handlers for btnLoginA and btnLoginB. You’ll notice that, as with the single- and multi-line input controls, to obtain the text from the Web Control the Text property is used, and with the Html Control the Value property is used.

Listing 5.13

```
Private Sub btnLoginA_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnLoginA.Click
  If txtUsernameA.Text = "Peter" And txtPasswordA.Text = "MyPassword"
Then
    lblResultsA.Text = "Access granted."
  Else
    lblResultsA.Text = "Access denied."
  End If
End Sub

Private Sub btnLoginB_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnLoginB.Click
  If txtUsernameB.Text = "Peter" And txtPasswordB.Value = "MyPassword"
Then
    lblResultsB.Text = "Access granted."
  Else
    lblResultsB.Text = "Access denied."
  End If
End Sub
```

The code is very simple, and does exactly the same thing for both buttons. It starts off with an If statement saying that ‘if the username entered was Peter and the password entered was MyPassword then’. If this evaluates true, then the respective label’s Text property is set to “Access granted.”, and in all other cases it is set to “Access denied.”

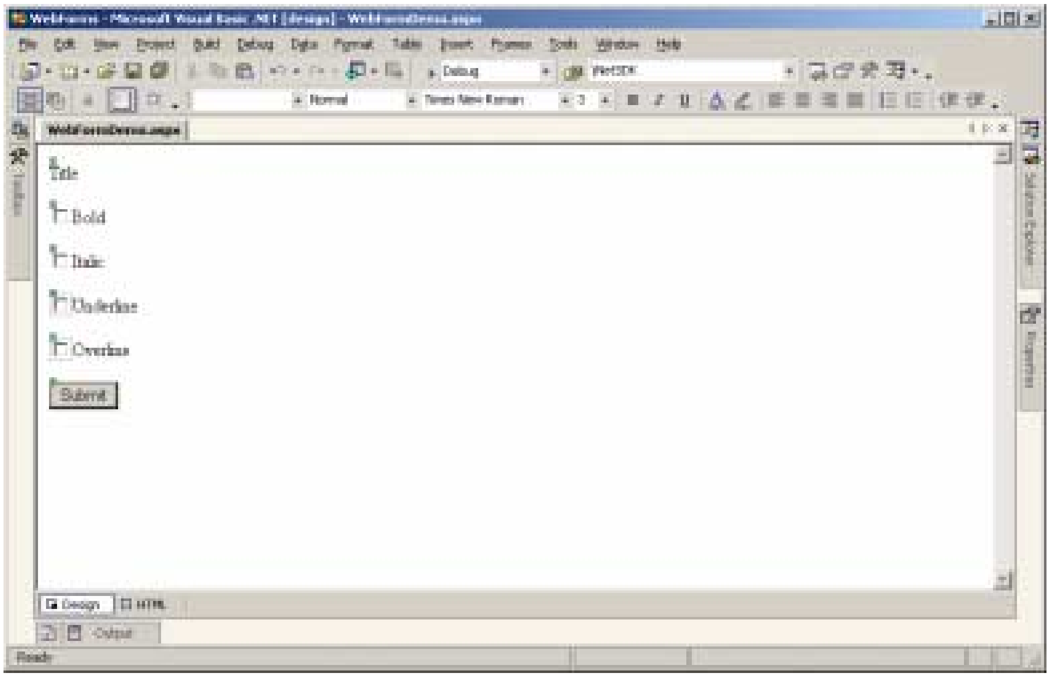
> Obviously this isn’t how you’d implement a ‘real’ login, which entails quite a lot more effort – this simply demonstrates the use of the password field, which is a small part of building a login and authentication system.

The Checkbox

The CheckBox control allows the user to provide a yes or no (checked or unchecked) input, and is most often used when the user should be able to choose one or more options from a list pertaining to a specific topic. For example, if the user is asked to choose his/her favorite food(s) from a list, and should be able to choose more than one, then using CheckBox controls would be a possibility for handling the input, since each type of food (e.g. Steak, chops) could have its own CheckBox, and the user would be able to check each type of food that they like.

The Web Control and Html Control varieties of the CheckBox are very similar, although there are a few minor differences. To demonstrate the two, we’re going to build a page that allows you to modify the formatting of text in a Label Web Control. In the designer, add a Label, lblTitle, and set its Text property to “Title”. Below it, add two CheckBox Web Controls, underneath each other. Set the first one’s ID to chkBold and the second’s to chkItalic, and set their Text properties to “Bold” and “Italic” respectively. Next, add two CheckBox Html Controls, also underneath each other, and set their IDs to chkUnderline and chkOverline respectively. The Html Control CheckBox doesn’t have a Text property, so to add a caption to the controls you need to move the cursor to the right of the control and type the caption there. Give the first control the caption, “Underline” and the second one, “Overline”. Finally, add a Button Web Control, btnSubmit, with Text property “Submit”. Figure 5.22 shows how the form should now look in the designer.

Figure 5.22 – CheckBox controls in the designer


WebForms - Microsoft Visual Basic .NET [design] - WebFormsDemo.aspx
File Edit View Project Build Debug Data Format Table Insert Frames Tools Window Help
Debug
WebSDK
Normal
Times New Roman
WebFormsDemo.aspx
Title
Bold
Italic
Underline
Overline
Submit
Design
HTML
Output
Ready

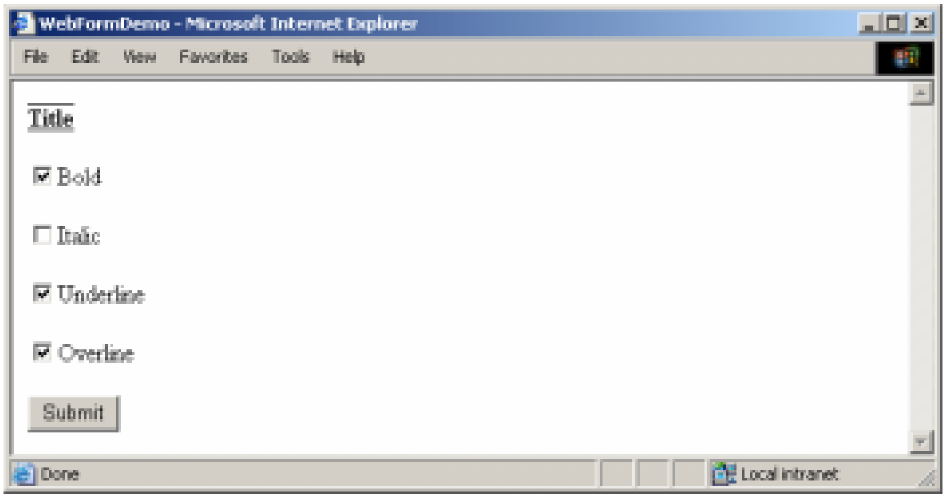
Double click the Button to create the Click event handler for it. In this event handler, the code must update the Label according to which options have been chosen for the formatting with the CheckBox controls. Take note that all of the options are Boolean properties, so if the corresponding CheckBox for a property is checked (i.e. True), then that property of the Label must be set to true. Because of this, implementing the functionality for the page is actually very simple, as shown in Listing 5.14.

Listing 5.14

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
  lblTitle.Font.Bold = chkBold.Checked
  lblTitle.Font.Italic = chkItalic.Checked
  lblTitle.Font.Underline = chkUnderline.Checked
  lblTitle.Font.Overline = chkOverline.Checked
End Sub
```

Both the Web Control and Html Control CheckBoxes expose a “Checked” property, of type Boolean, that is True if the CheckBox is checked, and False if it is not. Therefore in the first line, if chkBold is checked, then the Label’s Bold property is set to True, and if it isn’t then it is set to False, because the chkBold.Checked property would be false. Figure 5.23 shows the page in action.

Figure 5.23 – CheckBox controls manipulating the Font property of a Label


WebFormDemo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Submit
Done
Local intranet

Title

☑ Bold

☐ Italic

☑ Underline

☑ Overline

AutoPostBack and the CheckedChanged and ServerChange Events

One of the features that many Web Controls have that Html Controls do not is the ability for the control to automatically cause a postback (i.e. submit the form) when it is changed. In fact, we’ve actually already covered a control that has this functionality – the TextBox Web Control. However, it is not normally used with TextBox controls as there aren’t very many constructive uses for this feature with them, but when specific choices are involved, such as with CheckBoxes, as well as Radio Buttons and DropDowns, automatic postbacks can prove to be very useful.

To demonstrate the AutoPostBack, we’re going to make the page automatically update itself every time one of the Web Control CheckBoxes is changed. Firstly, to enable AutoPostBack, set the AutoPostBack property of both Web Control CheckBoxes to True. Since the Button isn’t going to be clicked, its Click event isn’t going to be raised, so the code to make adjustments to the Label’s formatting must go elsewhere. The CheckBox control exposes a CheckedChanged event, which fires during a postback if the Checked property of the control has changed since the last time the page was loaded. To create an event handler for the CheckedChanged event for the first CheckBox, simply double click it in the designer. This event handler only pertains to the chkBold CheckBox, so insert the following line of code in it:

```
lblTitle.Font.Bold = chkBold.Checked
```

Switch back to the designer and double click on chkItalic, and place the following line in the event handler that is created for its CheckedChanged event:

```
lblTitle.Font.Italic = chkItalic.Checked
```

Now when you load the page in IE and click either of those two CheckBoxes, you’ll notice that the page automatically ‘posts back’, and the changes are visible in the Label without having to cause a postback by clicking the Button.

The AutoPostBack functionality is particularly useful in longer forms where a part of the form might rely on a choice that is made further up. For example, a form might exist where the user is asked to choose his/her favorite type(s) of magazine (e.g. Current Affairs; Humour; Lifestyle etc.), and then based on what they chose there, they could be asked what magazines they purchase regularly later on, with the choices of magazines limited to those in the genres that they selected. You've probably come across this before during your travels on the web, but with ASP.NET, all the hard work is done for you, so implementing a form like this isn't difficult at all, as will be shown with the next control to be overviewed, the CheckBoxList.

However, before you decide to implement all your forms using Web Controls with AutoPostBack enabled, bare in mind the performance cost of the postback to the user – the user doesn't enjoy waiting for pages to load at the best of times, so you shouldn't use the AutoPostBack feature unless you're actually going to add enough value to what the user is doing to offset the performance hit. If using an AutoPostBack is going to make the user experience better, go for it by all means, but don't use AutoPostBacks needlessly where they're not really essential and will only frustrate users with slow connections.

Even though the CheckBox Html Control doesn't provide AutoPostBack functionality, it does have an event that fires during postbacks if it has changed since the last postback, namely the ServerChange event. Likewise, even though the CheckBox Web Control can automatically post the page back to the server when it is changed, it doesn't have to (simply leave the AutoPostBack property set to False), but its CheckedChanged event will still fire during a postback if the control's Checked property has changed, even though AutoPostBack is not enabled.

To demonstrate the Changed events on both the Html and Web Controls, without AutoPostBack, start by reverting the AutoPostBack property on chkBold and chkItalic back to False. The CheckedChanged event handlers for those two controls have already been created, so move on to the two Html Controls. Double click on chkUnderline, and an event handler for the ServerChange event should be created. Insert the following line into it:

```
lblTitle.Font.Underline = chkUnderline.Checked
```

Switch back to the designer and create a ServerChange event handler for chkOverline and insert the following line:

```
lblTitle.Font.Overline = chkOverline.Checked
```

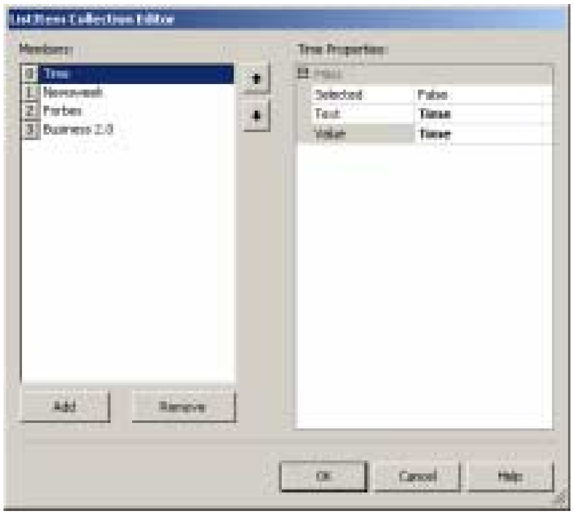
Now that all the CheckBox controls are handling the formatting that they are responsible for in the Label, there's no need for the Button's Click event handler, so you can either delete the contents of the event handler, or the entire subroutine itself. When you load the page, you'll find that it now operates identically to how it did originally – proof, if it was ever needed, that there's almost always more than one way to do something in programming! There is actually a slight difference between the two that doesn't effect the user, but exists nonetheless though – using the first method, every time the Button is clicked, all of the 4 font properties on the Label are assigned, even though they may not necessarily change. However, using the "Change event" method, a font property is only ever assigned when the corresponding CheckBox is changed, because the CheckedChanged and ServerChange events only fire when it does.

The CheckBoxList

The CheckBoxList is a Web Control (and doesn't have an Html Control implementation, since it doesn't exist as a pure HTML element) that displays a list of checkboxes. The power of this control is that it allows you to programmatically access the list, so you can dynamically add, edit or remove checkboxes from the list, as well as being able to loop through them, manipulating each checkbox item as you go along. This provides a great deal of flexibility and is especially useful when it is necessary to change the options available to the user dynamically, according to other choices that they have made previously.

We'll start off by creating a CheckBoxList with several options manually at design-time. To do this, place a CheckBoxList control onto the page, and set its ID to lstMagazines. The control will be used to display a list of magazines, and the user will be able to select which ones he/she purchases regularly. To add checkboxes to the list, select the Items property in the Properties Window, and click on the button to the right of it that appears when the property is selected. The ListItem Collection Editor should appear. Click "Add" to add a new item to the list, and in the right pane, you can choose whether the checkbox item will be selected by default, and provide the Text and Value properties for it. Leaving the Value as the same as the Text property is perfectly adequate (once you move on to creating a new item, the editor will automatically fill in the Value to be the same as the Text if you didn't enter a value into it), unless you have a specific reason for not doing so. However, when using the CheckBoxList where the choices are retrieved from a database, you will normally use the Value to store the identity (ID) field from the database table. Figure 5.24 shows the Collection Editor after a few items have been added.

Figure 5.24 – The ListItem Collection Editor



0 Time
1 Newsweek
2 Forbes
3 Business 2.0

Selected False
Text Time
Value Time

Using the Up and Down arrows to the right of the left pane, you can modify the order of the checkboxes, and you can remove the selected item by clicking on the "Remove" button. Once you have finished adding items, click on the "OK" button to close the editor. You will now notice that the CheckBoxList on the page has been updated to show the checkboxes that you have entered in the Collection Editor.

Add a Button Web Control, btnSubmit, to the page, set its Text property to "Submit" and place it below lstMagazines. Below it, place a Label Web Control, lblSelected, with its Text property set to nothing. When the Button is clicked, the Label will show a list of all the items that are selected in lstMagazines. To do this, create a Click event handler for btnSubmit by double clicking it, and insert the code in listing 5.15.

Listing 5.15

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
  Dim strSelectedOptions As String = ""
  Dim i As Integer
  For i = 0 To lstMagazines.Items.Count - 1
    If lstMagazines.Items(i).Selected Then
      If strSelectedOptions <> "" Then
        strSelectedOptions = strSelectedOptions & ", "
      End If
      strSelectedOptions = strSelectedOptions & lstMagazines.Items(i).Text
    End If
  Next
  If strSelectedOptions = "" Then
    lblSelected.Text = "No options selected."
  Else
    lblSelected.Text = "Selected options: " & strSelectedOptions
  End If
End Sub
```

Before assigning any value to the Label, the procedure stores the list of selected items in a String variable, strSelectedOptions, declared in the first line. The crux of the procedure is in the For loop. The loop iterates through all the items in the list – take note that the Items.Count property returns the number of items in the list, but the indexes of the items in the list are 0-based, so 1 is subtracted. The Items property of the CheckListBox controls contains the collection of the items in the list (a collection of System.Web.UI.WebControls.ListItem objects), and it is possible to access an individual item in the list by using the statement CheckBoxList.Items(itemindex). Assuming that a corresponding ListItem for the itemindex passed into the paramatized Items property exists, this statement will return a reference to a ListItem object, which can then be manipulated. In the first If statement inside the loop, the current ListItem’s Selected property is inspected. If it is True, then the corresponding checkbox must be checked, and so be added to the string containing the list of selected options, strSelectionOptions. However, if it is not selected, nothing must happen, and the loop must simply continue to the next iteration. The following code appears inside the If block.

```
If strSelectedOptions <> "" Then
  strSelectedOptions = strSelectedOptions & ", "
End If
strSelectedOptions = strSelectedOptions & lstMagazines.Items(i).Text
```

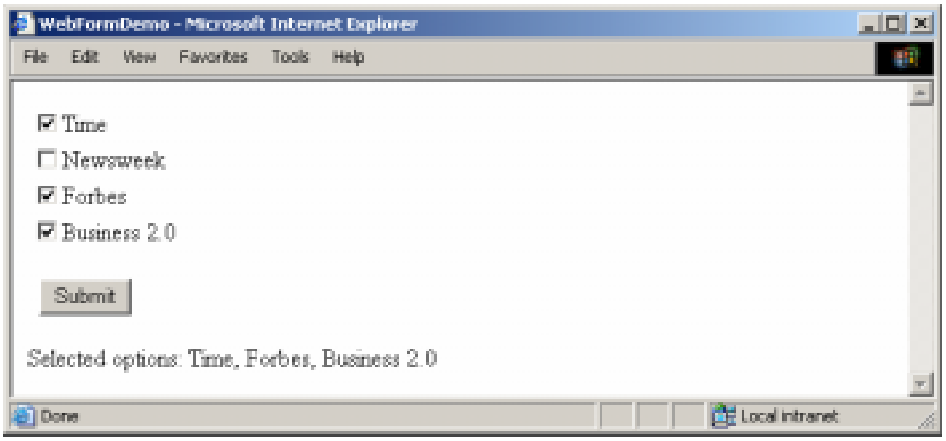
The If statement is simply there to ensure that the list of selected options is comma-separated. The first selected option must obviously not have a comma in front of it, so when strSelectedOptions is empty, a comma must not be added. However, if there is text in strSelectedOptions, a comma should be added.

Finally, the ListItem’s Text property is used to retrieve the Text from the checkbox and this is appended to strSelectedOptions.

The code to update the display of lblSelected follows the loop. If strSelectedOptions is empty, then no options are selected, so a message to that effect is displayed. However, if strSelectedOptions contains text, then at least one option was selected, so strSelectedOptions must be outputted to the Label.

To test out the code, load the page in IE, select a few options and click the “Submit” button. Figure 5.25 shows sample output from the page.

Figure 5.25 – The CheckBoxList in action



☑ Time
□ Newsweek
☑ Forbes
☑ Business 2.0

Selected options: Time, Forbes, Business 2.0

Before we move on to adding or removing items from the list programmatically, I’ll demonstrate another way to write the same procedure for the Button Click event handler. Those of you more familiar with the Visual Basic language will probably have noticed that the above scenario looks like the perfect place to use a For Each loop, and you’d be correct. There is no “better” method to iterate through the list items, as both the original For loop, and the For Each loop that I’m about to show do exactly the same job, but it is useful to know about both methods, as there are examples throughout the book that demonstrate concepts using one or the other.

Listing 5.16 shows the revised Click event handler that replaces the For loop with a slightly different variant that doesn’t make use of a counter variant (i).

Listing 5.16

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
  Dim strSelectedOptions As String = ""
  Dim Item As ListItem
  For Each Item In lstMagazines.Items
    If Item.Selected Then
      If strSelectedOptions <> "" Then
        strSelectedOptions = strSelectedOptions & ", "
      End If
      strSelectedOptions = strSelectedOptions & Item.Text
    End If
  Next
  If strSelectedOptions = "" Then
    lblSelected.Text = "No options selected."
  Else
    lblSelected.Text = "Selected options: " & strSelectedOptions
  End If
End Sub
```

The For Each loop is a native VB.NET language structure that is designed specifically for iterating through collections. In the “declaration” of the loop,

```
  For Each Item In lstMagazines.Items
```

Item represents a single item in the collection that is being looped through, and the object after “In”, lstMagazines.Items, is that collection. For each iteration of the loop, the Item object is reassigned as a reference to the current item in the collection. Because of this, the code inside the loop uses the Item object to obtain information about the current ListItem.

If you load the page again, you’ll find that it operates exactly the same as before, as the code is functionally identical.

Adding and Removing ListItems Programmatically

As was mentioned at the beginning, one of the more useful aspects of this control is that it is simple to add and remove items to and from it dynamically, during run-time. We’ll start off by simply allowing the user to add his/her own choices to the magazine list, and because the existing code to report which options are selected iterates through all the items using a loop, the new options will automatically be recognized in the code.

In the Design view, above lstMagazines, add a TextBox Web Control, txtNew, and a Button Web Control, btnAdd, and set its Text property to “Add”. The user will type in the name of the item to add in the TextBox, and when the “Add” button is pressed, it will be added to the list. To implement this, double click on btnAdd to create a Click event handler for it, and use the code shown in Figure 5.17.

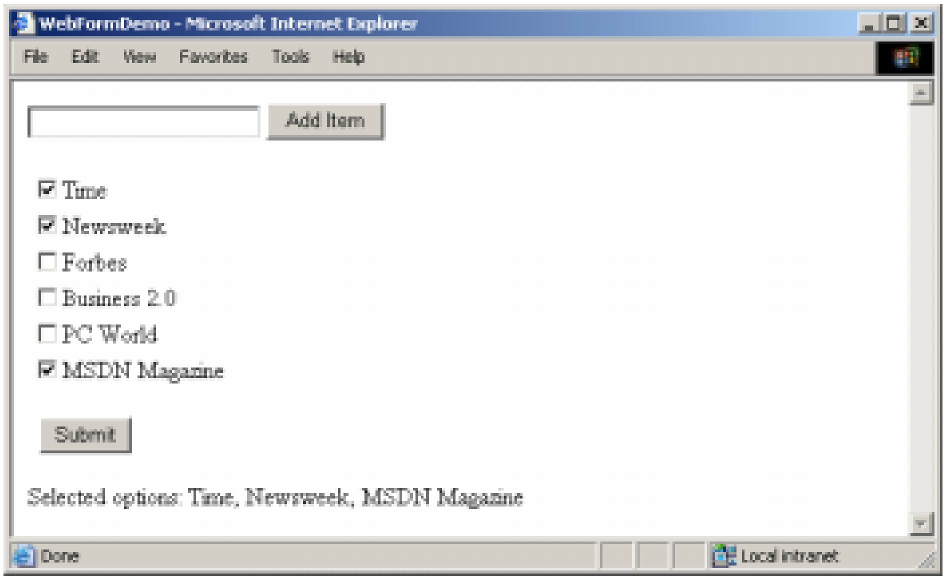
Listing 5.17

```
  Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As
  System.EventArgs) Handles btnAdd.Click
    Dim NewItem As New ListItem()
    NewItem.Text = txtNew.Text
    lstMagazines.Items.Add(NewItem)
    txtNew.Text = ""
  End Sub
```

The first line creates a new ListItem object, NewItem. Next, the Text property of NewItem is set, and if necessary other properties, such as Selected (which is False by default), and Value, could also be set. After setting the property(ies) on the NewItem object, it is added to the lstMagazines CheckBoxList through the Items’ property’s Add method. The Add method accepts a ListItem object as a parameter (although there is another overloaded version of it that accepts a string and creates the ListItem for you, this version is more flexible), and adds the ListItem to the collection. Finally, the txtNew TextBox is cleared.

As was mentioned earlier, since the “Submit” Button’s event handler loops through all the items in the CheckBoxList, any additions to the list will be included. Figure 5.26 shows the page after several extra options have been added and the Submit button has been clicked.

Figure 5.26 – Dynamically adding items to a CheckBoxList



☑ Time
☑ Newsweek
□ Forbes
□ Business 2.0
□ PC World
☑ MSDN Magazine

Selected options: Time, Newsweek, MSDN Magazine

Removing an item can actually be slightly trickier than adding one, depending on how you do it. When adding an item, you simply provide the text for it and call the Add method. However, when removing an item, you obviously have to know which item you want to remove from the list, and the item might be the first in the list, or the last – and you might not know its exact position. For this reason, the CheckBoxList control gives you a choice – you can either remove an item according to its “index” (its position in the list, keeping in mind the fact that the first item in the list is 0), or you can remove an item according to a reference to the actual ListItem object, or simply the text that the item displays.

First off, add a Button Web Control next to the “Add Item” button, and call it btnDelete. Set its Text property appropriately and create a Click event handler for it. Listing 5.18 shows the code for the event handler that removes the item named in the TextBox.

Listing 5.18

```
Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnDelete.Click
  lstMagazines.Items.Remove(txtNew.Text)
  txtNew.Text = ""
End Sub
```

When running the page, to remove an item, simply type in its name (case-sensitive) in the TextBox and click the “Delete Item” button.

The SelectedIndexChanged Event

As with the CheckBox, and many other Web Controls, the CheckBoxList provides an event that fires when the user modifies it. In the case of the CheckBoxList, this event is SelectedIndexChanged, and fires during a postback if any of the ListItems (checkboxes) Selected properties have changed since the last postback. The CheckBoxList also has an AutoPostBack property, which causes an immediate postback when any of the checkboxes in the list are clicked. In the final section of this introduction to the CheckBoxList control, we'll make use of both these features.

Building a Dynamic CheckBoxList Form

We're going to modify the current form, because it's not a very realistic representation of something you might do in a real application. The user will almost never be responsible for manually adding or removing items to and from the CheckListBox, so instead of the TextBox and the two Button controls at the top, we'll have several CheckBox controls, providing different categories of magazines. When one of these CheckBox controls is checked, the CheckBoxList must show options in that category of magazine, and if the user unchecks the control, the related options must be removed from the CheckBoxList. Although its not necessary, and is often not good practice, in the spirit of demonstrating what the CheckBoxList is capable of, we'll also make use of its AutoPostBack functionality, as well as that of the CheckBox controls, to ensure that the Label control listing the currently-selected options is always up-to-date.

First up, we want to move the code that is currently in btnSubmit's Click event handler to a separate, standalone subroutine. To do this, insert the following code into the codebehind file, preferably just before the End Class statement:

```
Private Sub UpdateLabel()


End Sub
```

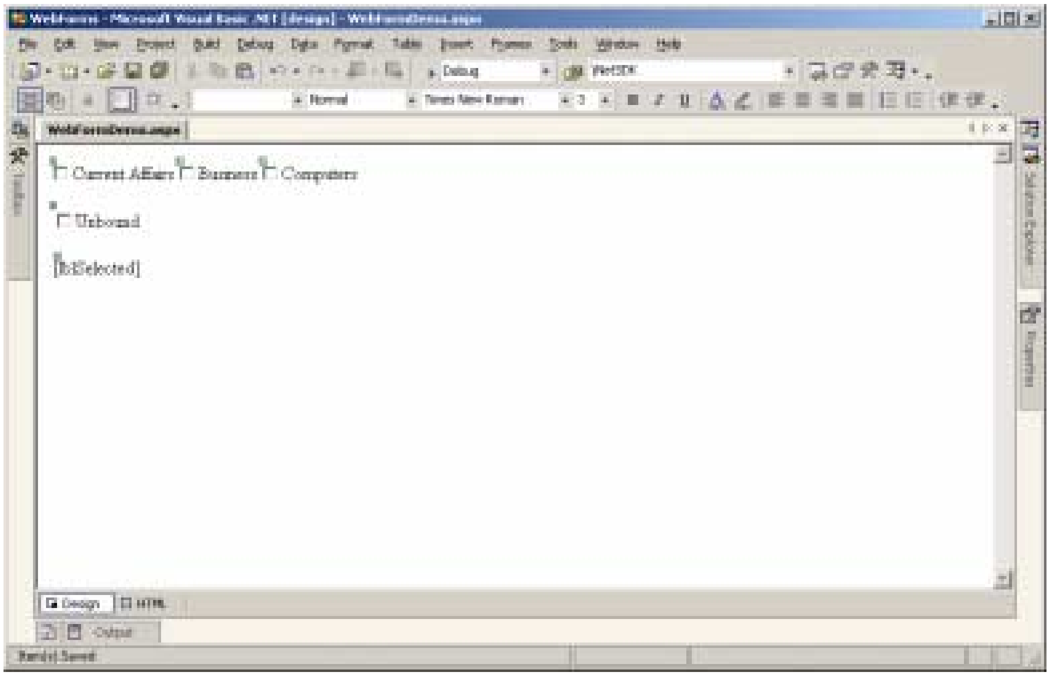
Now copy and paste the code inside the btnSubmit Click event handler (but not including the Private Sub and End Sub statements) into the newly created UpdateLabel subroutine. This procedure will be called from numerous different event handlers, so instead of duplicating the code, we've made an entirely separate procedure that can be called when needed. Since we'll be using AutoPostBack you can delete the entire btnSubmit Click event handler (the btnSubmit_Click subroutine), and then switch to Design mode and delete the btnSubmit Button. Select lstMagazines and set its AutoPostBack property to True. Double-click on the control (lstMagazines), and the IDE should switch over to the server-side code and have created a new event handler for the SelectedIndexChanged event. As has already been discussed, this event fires when an item in the list changes (because it is clicked), so therefore the Label displaying the list of currently selected options must be updated. To do this, simply place the following line into the event handler:

```
UpdateLabel()
```

With this done, you can now load the page in IE and try out the AutoPostBack functionality of the CheckBoxList. The form operates similarly to before, except instead of clicking the "Submit" button on the form to update the Label, the form submits itself every time a checkbox is clicked, and the Label is then updated automatically.

With that out of the way, we can progress with the more interesting parts. First off, switch to the server-side code and remove the Click event handlers for the btnAdd and btnDelete Button controls. Switch back to the Design mode and remove the TextBox and the Button controls, and in their place, put three CheckBox controls. Set the first ones ID property to chkCurrentAffairs, and set its Text property to “Current Affairs”. Do similarly with the next two controls, but with the IDs chkBusiness and chkComputers, and the Text, “Business” and “Computers” respectively. Set the AutoPostBack property of all three to True. Select lstMagazines and remove all the current items using the Items property and Collection Editor. The form in the Designer should now look similar to Figure 5.27.

Figure 5.27 – A form using both CheckBox controls and a CheckBoxList


WebForms - Microsoft Visual Basic .NET [design] - WebFormsDemo.aspx
File Edit View Project Build Debug Data Format Table Insert Frames Tools Window Help
Debug
WebSDK
Normal
Times New Roman
3
WebFormsDemo.aspx
□ Current Affairs □ Business □ Computers
□ Unbound
[lblSelected]
Design
HTML
Output
Ready Saved

The final code that still needs to be implemented is that for the three CheckBox controls’ CheckedChanged events. Starting with chkCurrentAffairs, when its CheckedChanged event fires, items must either be added to or removed from lstMagazines. If chkCurrentAffairs.Checked is True, then items, such as “Time” and “Newsweek” must be added because the checkbox has just been checked. However, if it is False, then these items must be removed. Finally, to ensure that the lblSelected Label is updated, the UpdateLabel procedure must be called.

Create CheckedChanged event handlers for all three CheckBox controls by double clicking them, and insert the code in listing 5.19.

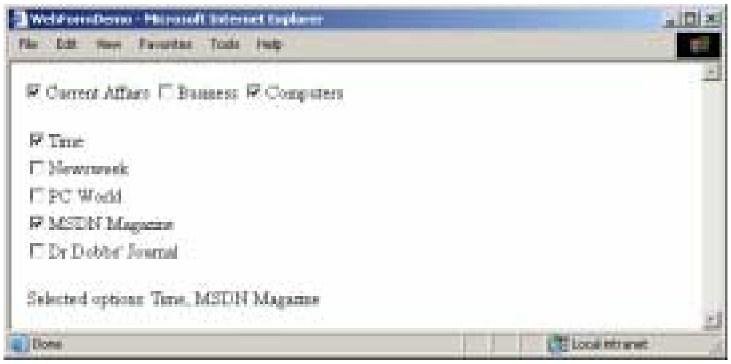
Listing 5.19

```
Private Sub chkCurrentAffairs_CheckedChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
chkCurrentAffairs.CheckedChanged
  If chkCurrentAffairs.Checked Then
```

```
    lstMagazines.Items.Add("Time")
    lstMagazines.Items.Add("Newsweek")
  Else
    lstMagazines.Items.Remove("Time")
    lstMagazines.Items.Remove("Newsweek")
  End If
  UpdateLabel()
End Sub

Private Sub chkBusiness_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles chkBusiness.CheckedChanged
  If chkBusiness.Checked Then
    lstMagazines.Items.Add("Forbes")
    lstMagazines.Items.Add("Fortune")
    lstMagazines.Items.Add("Business 2.0")
  Else
    lstMagazines.Items.Remove("Forbes")
    lstMagazines.Items.Remove("Fortune")
    lstMagazines.Items.Remove("Business 2.0")
  End If
  UpdateLabel()
End Sub

Private Sub chkComputers_CheckedChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
chkComputers.CheckedChanged
  If chkComputers.Checked Then
    lstMagazines.Items.Add("PC World")
    lstMagazines.Items.Add("MSDN Magazine")
    lstMagazines.Items.Add("Dr Dobbs' Journal")
  Else
    lstMagazines.Items.Remove("PC World")
    lstMagazines.Items.Remove("MSDN Magazine")
    lstMagazines.Items.Remove("Dr Dobbs' Journal")
  End If
  UpdateLabel()
End Sub
```

Each event handler does essential the same thing – it checks whether the CheckBox control that is handles is checked. If so, it adds items to the CheckBoxList. If not, then it removes the items. Finally, UpdateLabel is called, in case items that were selected were removed. Figure 5.28 shows the final page after its been loaded in IE.

Figure 5.28 – Dynamically creating and removing CheckBoxList items


WebFormsDemo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
Local intranet

☑ Current Affairs ☐ Business ☑ Computers

☑ Time
☐ Newsweek
☐ PC World
☑ MSDN Magazine
☐ Dr Dobbs' Journal

Selected options: Time, MSDN Magazine

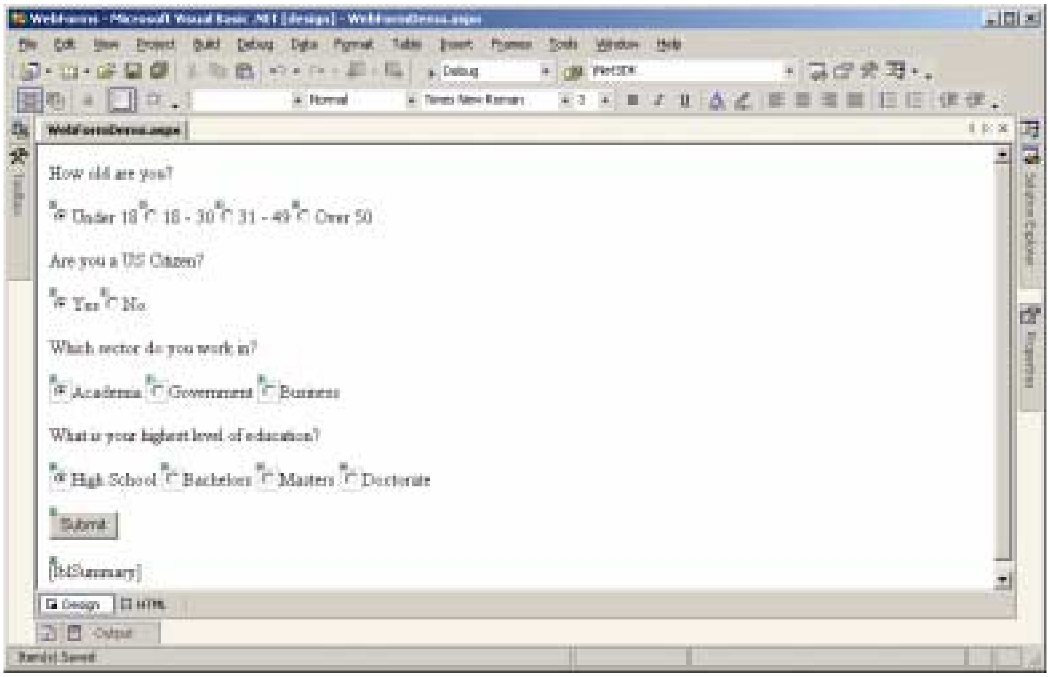
The Radio Button

The RadioButton control shares several similarities with the CheckBox control, starting with the fact that both controls come in both the Web Control and Html Control varieties. Similar to the CheckBox, the RadioButton's functionality also revolves around providing the user with a choice from numerous options, and exists in either a checked or unchecked state. However, whilst the CheckBox control allows the user to select (i.e. check) more that one option, the RadioButton control exists for scenarios where only one option can be selected at a time. For example, when asking the user which age group they fit into, you would not use CheckBox controls to provide the options "Under 18", "18 – 25", "26 – 35" etc, because only one of the options could possibly be valid, so RadioButton controls would be used.

The RadioButton control is very easy to use, and is implemented very similarly to the CheckBox. However, while the CheckBox doesn't have any mandatory properties, so to speak (the ID property might be considered mandatory, but even if you leave it at its default value, the functionality of the CheckBox won't be hindered), the RadioButton control, in both the Web Control and Html Control implementations, has one property in each variety that must be set in order for the controls to function correctly. This is the GroupName property, in the case of the Web Control, and the name property for the Html Control. RadioButton controls must be "grouped" together, so that all the options that are related to a specific question are explicitly "connected". This is very simple to do, and all that is required is that all the RadioButton controls that related to a specific question must all have the same GroupName property value (or name property value, if Html Control RadioButtons are being used). This is a string property, so for a question relating to the age of the user, the GroupName (or name) property could be set to "Age" for all the RadioButton controls that provide options for the answer. If you ever have problems with RadioButton controls, where either the user can select more than one option for a particular question, or the user can only select one option between two (or more!) questions, then it's more than likely that you've either forgotten to set the GroupName or name properties, or have set them incorrectly.

With the theory out of the way, we can move onto actually using the RadioButton controls. To demonstrate the two different types (Web and Html), and the "grouping" functionality for each, we'll set up a questionnaire containing 4 questions, two using Web Control RadioButtons for the options and two using Html Control RadioButtons. The form should look like Figure 5.29 in the Designer.

Figure 5.29 – RadioButton controls in the Designer



WebForms - Microsoft Visual Basic .NET [design] - WebFormsDemo.aspx
File Edit View Project Build Debug Data Format Table Insert Frames Tools Window Help
Debug
WebSDK
Normal
Times New Roman
WebFormsDemo.aspx
Toolbox
Solution Explorer
Properties
Design
HTML
Output
Ready Saved

How old are you?

● Under 18 ○ 18 - 30 ○ 31 - 49 ○ Over 50

Are you a US Citizen?

● Yes ○ No

Which sector do you work in?

● Academia ○ Government ○ Business

What is your highest level of education?

● High School ○ Bachelors ○ Masters ○ Doctorate

Submit

[lblSummary]

The first two questions use Web Controls, and the second two use Html Controls. As with the CheckBox controls, the Web Controls provide a Text property for the caption accompanying each option, but the Html Controls do not, so you must enter the text in the Designer. Add the required question text and all the RadioButton controls to the page, and set the Text property, or add the text in the Designer, for each RadioButton using the caption text shown in the figure. Set the ID properties for the RadioButton controls as follows:

For the first question, rdoAgeUnder18, rdoAge18to30, rdo31to49 and rdoOver50.

For the second question, rdoCitizenYes, rdoCitizenNo.

For the third question, rdoSectorAcademia, rdoSectorGovernment, rdoSectorBusiness.

For the fourth question, rdoEducationHighSchool, rdoEducationBachelors, rdoEducationMasters, rdoEducationDoctorate.

Take note of the structure of the IDs. Although obviously not mandatory, it is good practice to prefix the ID of each control that are options for the same question with the same string, so for the first question, which deals with age, the prefix is “rdoAge”. This helps avoid possible confusion later on when programming the server-side code as it helps to reinforce what options “belong” to which questions.

After setting the ID and Text properties, the final property to set is the GroupName or name property. For the options for the first question, set each control’s GroupName property to “Age”, and for the second question’s options, set it to “Citizen”. The thirds and fourth questions use Html Controls, so for the third question, set the name property (not ID, but name) of each control to “Sector”, and for the final question, set it to “Education”.

It is possible to set the value of a property on numerous controls at the same time by selected all the controls and then changing the property in the Properties window. To do this, hold down the Ctrl key whilst clicking controls, and you'll see that each control is simply added to the selection. Once all the controls you want are selected, you can modify their properties all at once. This technique also works for converting HTML controls to Server Controls – simply select all the controls you want to convert, and then right click one of them and choose "Run As Server Control".

Another property that you may want to set on each of the first RadioButton controls for each question is Checked, setting it to True to make the first option of each question selected.

With all the RadioButton controls set up, you can add a Button Web Control, btnSubmit (with its Text property set to "Submit"), and a Label, lblSummary, beneath it, with its Text property set to nothing. When btnSubmit is clicked, the Label will display the chosen options of the user. To implement this, double click btnSubmit in the Designer to create an event handler for its Click event, and use the code in listing 5.20.

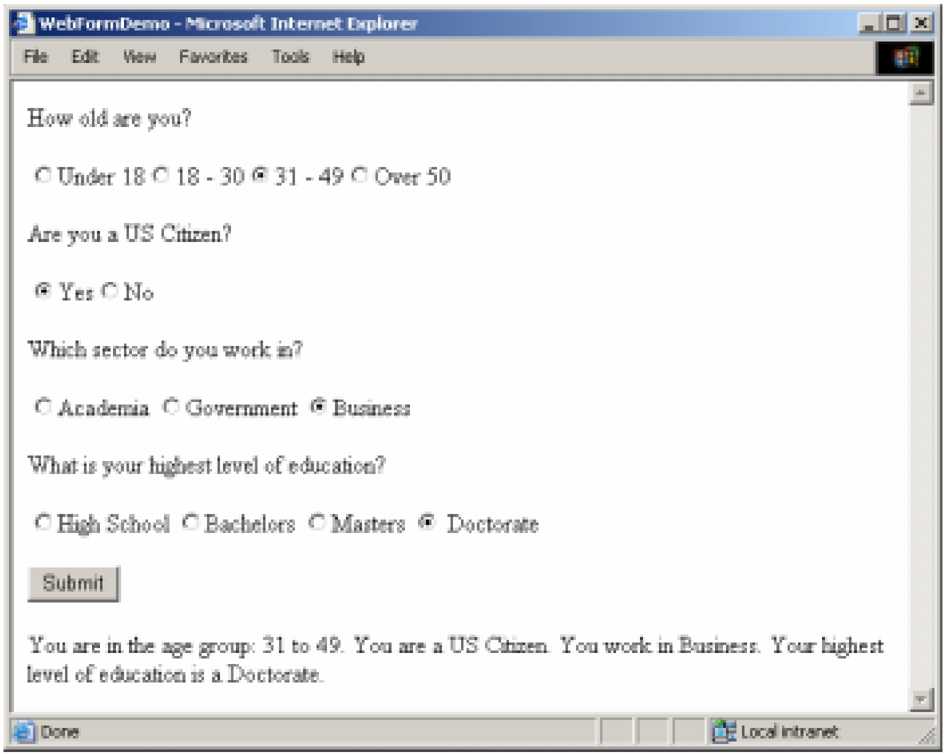
Listing 5.20

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
  Dim objStringBuilder As New System.Text.StringBuilder()
  Dim strTemp As String
  objStringBuilder.Append("You are in the age group: ")
  If rdoAgeUnder18.Checked Then
    strTemp = "Under 18"
  ElseIf rdoAge18to30.Checked Then
    strTemp = "18 to 30"
  ElseIf rdoAge31to49.Checked Then
    strTemp = "31 to 49"
  ElseIf rdoAgeOver50.Checked Then
    strTemp = "Over 50"
  End If
  objStringBuilder.Append(strTemp)
  objStringBuilder.Append(". ")
  If rdoCitizenYes.Checked Then
    strTemp = "You are a US Citizen."
  Else
    strTemp = "You are not a US Citizen."
  End If
  objStringBuilder.Append(strTemp)
  objStringBuilder.Append(" You work in ")
  If rdoSectorAcademia.Checked Then
    strTemp = "the Academia"
  ElseIf rdoSectorGovernment.Checked Then
    strTemp = "Government"
  ElseIf rdoSectorBusiness.Checked Then
```

```
      strTemp = "Business"
    End If
    objStringBuilder.Append(strTemp)
    objStringBuilder.Append(". Your highest level of education is ")
    If rdoEducationHighSchool.Checked Then
      strTemp = "High School"
    ElseIf rdoEducationBachelors.Checked Then
      strTemp = "a Bachelor's degree"
    ElseIf rdoEducationMasters.Checked Then
      strTemp = "a Master's degree"
    ElseIf rdoEducationDoctorate.Checked Then
      strTemp = "a Doctorate"
    End If
    objStringBuilder.Append(strTemp)
    objStringBuilder.Append(".")
    lblSummary.Text = objStringBuilder.ToString()
  End Sub
```

While the listing is fairly long, it doesn’t include any particularly complex code, and clearly demonstrates the reason why the RadioButton control isn’t a good choice when numerous options are available to the user, as each one has to be coded individually on the server-side. The only RadioButton property that is used is the Checked property, which exists in both the Web and Html Control varieties, and is True when an option is checked, and False when it is not. However, to generate the summary string (the StringBuilder object, which was introduced earlier in the chapter, is used to create the string, since it involves numerous concatenations), each RadioButton has to have its Checked property inspected manually, as there is no easy way to loop through a specific “group” of options and return which option is selected. For this reason, I strongly recommend that you rather use the RadioButtonList control when you need to use radio buttons to provide options to the user, as this control, which is analogous to what the CheckBoxList is to a CheckBox, provides a programmable list of radio buttons, and provides properties and methods for quickly and easily obtaining the selected option. However, we’ll deal with that control in the next section. For now, figure 5.30 shows the page loaded in IE after the questionnaire has been answered and the form has been submitted.

Figure 5.30 – RadioButton Web and Html Controls being used in a questionnaire


WebFormDemo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Submit
Done
Local intranet

How old are you?

○ Under 18 ○ 18 - 30 ● 31 - 49 ○ Over 50

Are you a US Citizen?

● Yes ○ No

Which sector do you work in?

○ Academia ○ Government ● Business

What is your highest level of education?

○ High School ○ Bachelors ○ Masters ● Doctorate

You are in the age group: 31 to 49. You are a US Citizen. You work in Business. Your highest level of education is a Doctorate.

As with the CheckBox Web Control, the RadioButton Web Control also features an AutoPostBack property, so when the RadioButton either is checked, or becomes unchecked, a postback will be forced. Additionally, both the Web Control and Html Control varieties of the CheckBox provide events that trigger during a postback when their Checked properties have changed since the last postback. For the Web Control, the event in question is CheckedChanged, and for the Html Control it is ServerChange. In both cases, double-clicking on the control in the Designer will create the server-side event handler for the respective events.

The RadioButtonList

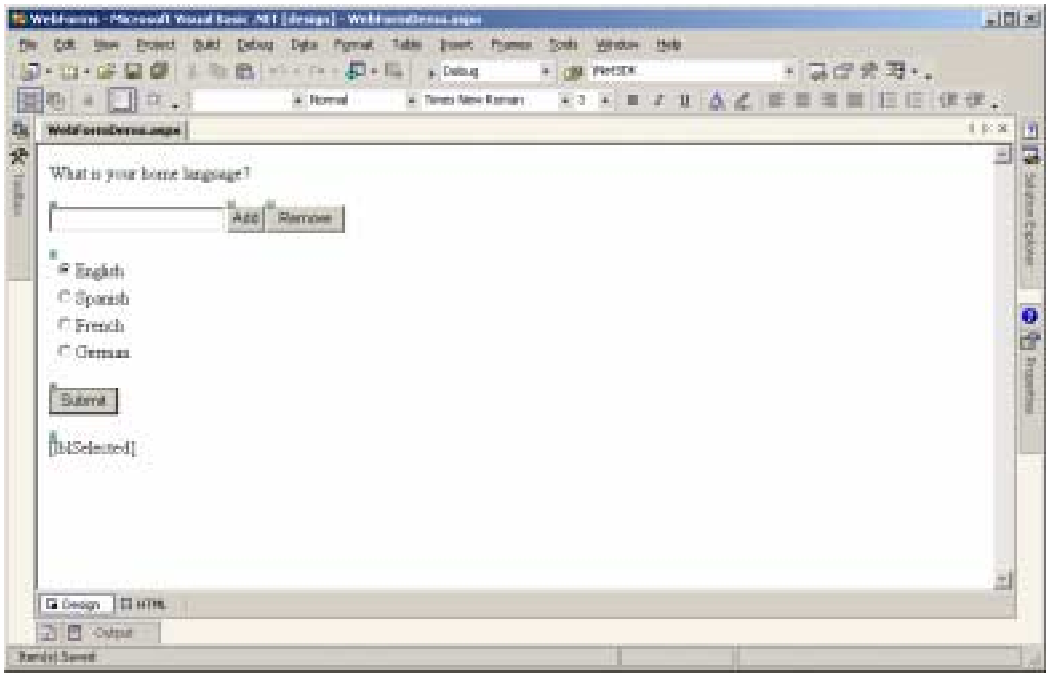
As was mentioned in the RadioButton section, the RadioButtonList provides a programmable interface to a list of radiobuttons in a very similar way to how the CheckBoxList contains a list of checkbox items and allows this list to be modified programmatically. In the RadioButton example, there was a significant amount of code to perform what should really be quite a simple operation – determining which RadioButton was selected. The RadioButtonList control greatly simplifies this process, and in fact reduces it to exactly one line of code. Unless you have a compelling reason to use several RadioButton controls instead of a RadioButtonList, I strongly recommend you avoid all the extra effort required by RadioButton controls and use the RadioButtonList.

Since the RadioButtonList is so similar to the CheckBoxList, which was introduced fairly thoroughly, we won't spend too much time on the RadioButtonList, as it should seem quite familiar, as there is very little that separates the two.

> The CheckBoxList and the RadioButtonList actually derive from the same base class, the ListControl class, so most of their functionality does actually overlap internally, and is why they expose such similar programmatic interfaces.

To demonstrate the RadioButtonList control, we'll create a form that performs the three major operations on it – adding new items, removing items, and determining which item is selected. Figure 5.31 shows how the form will look in the Designer.

Figure 5.31 – The RadioButtonList control



What is your home language?

Add Remove

● English
○ Spanish
○ French
○ German

Submit

[lblSelected]

Add a TextBox, txtNewLanguage and two Button controls, btnAdd and btnRemove. Following those three controls, add a RadioButtonList control, and set its ID to lstLanguage. As shown in Figure 5.31, it will be used to display a list of languages. Using the List Collection Editor, accessed via the Items property, add the items English, Spanish, French and German, and set the English item's Selected property to True. Finally add another Button, btnSubmit, and a Label, lblSelected. Set the Text property of all Buttons and the Label appropriately.

We'll ignore the Add and Remove buttons for the moment, and focus on implementing the event handler for the Submit button. When this button is clicked, the Label, lblSelected, should display the selected language. Listing 5.21 shows the event handler for btnSubmit's Click event.

Listing 5.21

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
  If Not IsNothing(lstLanguage.SelectedItem) Then
    lblSelected.Text = lstLanguage.SelectedItem.Text
  Else
    lblSelected.Text = ""
  End If
End Sub
```

Since the facility to remove items will be added, this event handler must cater that eventuality. Ordinarily, the only line needed would be the following:

```
lblSelected.Text = lstLanguage.SelectedItem.Text
```

The RadioButtonList object exposes a SelectedItem property, which returns a reference to the currently selected ListItem. This line therefore assigns the text from the currently selected radiobutton to the Label control. However, because items can be removed, a scenario might be created where there is no selected item, in which case the lstLanguage.SelectedItem property will return a null reference, to trying to access the Text property on it will cause an error to be raised. To avoid this, VB.NET's IsNothing function is first used to check that the lstLanguage.SelectedItem property does indeed contain an object (i.e. it isn't Nothing). The IsNothing function accepts any expression as its parameter and returns a Boolean value, and is very useful in circumstances such as these where you're not sure whether a variable contains an object instance or not.

The SelectedItem property is actually very useful, and if you consider how long the code in event handler would have to be if ordinary RadioButton controls were used, you'll really appreciate the power that it provides – instead of having a huge If block inspecting the value of each RadioButton's Checked property, you can simply call the SelectedItem property.

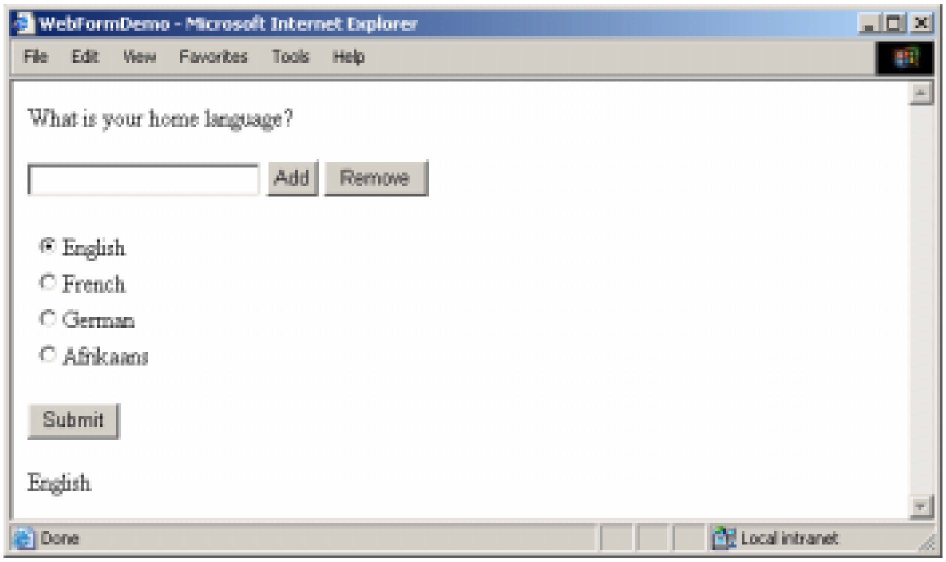
The two remaining pieces of implementation that still need attention are the Add Button's Click event handler, and the event handler for the Remove Button. The code is identical to the equivalent for the CheckBoxList, and is shown in Listing 5.22.

Listing 5.22

```
Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAdd.Click
  lstLanguage.Items.Add(txtNewLanguage.Text)
  txtNewLanguage.Text = ""
End Sub

Private Sub btnRemove_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRemove.Click
  lstLanguage.Items.Remove(txtNewLanguage.Text)
  txtNewLanguage.Text = ""
End Sub
```

In both cases the appropriate Items property method (Add and Remove respectively) is called, and the text in the TextBox is cleared. When the page is loaded, the user can add and remove items, and the selected item's text will be displayed in the Label when the Submit button is clicked. Figure 5.32 shows the page after the language Spanish has been removed and Afrikaans added, and English is selected.

Figure 5.32 – The RadioButtonList


WebFormDemo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Add
Remove
Submit
Done
Local intranet

What is your home language?

● English
○ French
○ German
○ Afrikaans

English

The ListBox and DropDownList

Both the ListBox and DropDownList controls have Web Control and Html Control implementations. Whilst the functionality between them is almost inseparable, there are a few differences between the Web Controls and the Html Controls when designing the page, which we will discuss shortly. In essence, the purpose of these two controls is to provide the user with a choice from a list of options. When the user needs to be presented with a list of choices, there are three controls that you could use – a RadioButtonList, a Web Control that doesn't have an Html Control equivalent; a ListBox Web or Html Control; or a DropDownList Web or Html Control. Which one you use depends largely on personal preference, but the number of options available should definitely be a factor in your decision. If there aren't very many options (less than 5), then the RadioButtonList and ListBox controls can be used, but if there are more than 5 options, then these two controls aren't very practical as they display all the options on the page, whereas the DropDownList only displays the currently selected option until its "dropdown menu" is opened to display the other options.

The ListBox and DropDownList Web Controls behave almost identically to the RadioButtonList both in the Designer, and during run-time. In the Designer, their list of options is modified using the ListItem Collection Editor, and during run-time their Items and SelectedItem property are used to add, edit and remove options and obtain information from the selected item in exactly the same way as is done with the RadioButtonList.

The Html Control varieties of the ListBox and DropDownList differ quite significantly from their Web Control equivalents in both design-time and run-time functionality. At design-time, these two controls do not have a ListItems Collection Editor, so the options for these controls must be entered manually in the HTML view, which simply makes them more tedious to use. During run-time, an Items property is exposed, so options can be added, edited and removed programmatically. However, the controls do not expose a SelectedItem property, which makes obtaining information about the selected option slightly more difficult than with the Web Controls, although it isn't a major issue.

To demonstrate the controls, we're going to create a page that allows several font properties of a Label Web Control to be set, including colors, fonts and font sizes. Figure 5.33 shows the final result in the Designer, so you know what you're working towards. To begin with, add a Label in the Designer, set its ID to lblTitle, and its Text property to "Title". We're going to organize our controls in a table, so next add a table to the form underneath the Label using the "Insert-Table" command from the "Table" menu. The table should have 5 rows and 2 columns. The first choice the user will have is that of the Label's font color; so in the first cell on the left hand side, enter a caption "Font Color". In the right-hand column, insert a DropDownList Web Control. Set its ID to lstFontColor, and using the Item property's ListItem Collection Editor, add four options – Black, Blue, Green and Red. The second choice will be of font size, so add an appropriate caption in the next row, and in the right column, insert a ListBox control, lstFontSize. Add the following options to it: XXSmall, Small, Medium, Large and XXLarge. Set the Small item's Selected property to True.

The Web Controls are quick and easy to set up. However, we're also going to demonstrate the ListBox and DropDown Html Controls, but these involve quite a bit more manual work to set up. To start off, a DropDown Html Control will be used to control the Label's Background Color, so in the 3rd table row, add an appropriate caption in the left column, and in the right column, add a DropDown with ID lstBackgroundColor, remembering of course to check its "Run As Server Control" option. To add items to the control at design time, you need to switch to HTML view, so do so. You need to find the line(s) of code that reads as follows.

```
<SELECT id=lstBackgroundColor name=Select1 runat="server">
<OPTION selected></OPTION>
</SELECT>
```

This is the declaration of the DropDown Html Control in the UI code. Take note of the runat attribute in the SELECT tag – this, as was discussed at the beginning of the chapter, signifies that this is a server control. To add options to the list, you use the same markup as if this were a regular HTML DropDown control – the OPTION tag. Each item needs to be declared using an OPTION tag in the following format:

```
<OPTION VALUE="value">text</OPTION>
```

A selected option has an additional attribute, and would be declared like this:

```
<OPTION VALUE="value" SELECTED>text</OPTION>
```

Add in the necessary OPTION tags so that the lstBackgroundColor UI code definition looks like listing 5.23:

Listing 5.23

```
<SELECT id=lstBackgroundColor name=Select1 runat="server">
 <OPTION value="White" selected>White</OPTION>
 <OPTION value="Blue">Blue</OPTION>
 <OPTION value="Red">Red</OPTION>
 <OPTION value="Green">Green</OPTION>
</SELECT>
```

With this done, you can now switch back to the Design mode and you should see that the control now shows that it does contain options. The final list control will be to change the font name, so add the appropriate caption to the last row, and in the right column, add a Listbox control from the HTML palette, setting it to run as a server control. Set its ID to lstFontName, and switch to the HTML view so that you can add items to it. Find the following line(s) of code:

```
<SELECT id=lstFontName size=2 name=Select1 runat="server">
<OPTION></OPTION>
</SELECT>
```

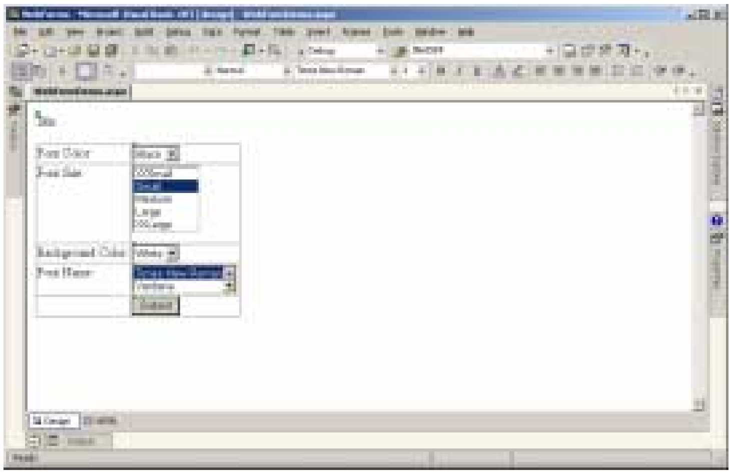
As you can see, the only attribute that differentiates a Listbox from a Dropdown Html Control is the inclusion of the size attribute in the SELECT tag. Since native HTML uses the SELECT tag for both dropdowns and listboxes, the Html Controls do to (since their real objective is to map as closely to native HTML as possible). Indeed, the class that is dealt with on the server-side is the same for both controls – the System.Web.UI.HtmlControls.HtmlSelect. Add the necessary OPTION elements so that the lstFontName definition is as shown in listing 5.24:

Listing 5.24

```
<SELECT id=lstFontName size=2 name=Select1 runat="server">
 <OPTION value="Arial">Arial</OPTION>
 <OPTION value="Courier New">Courier New</OPTION>
 <OPTION value="Times New Roman" selected>Times New
Roman</OPTION>
 <OPTION value="Verdana">Verdana</OPTION>
</SELECT>
```

You can now switch back to the design view and add a Button, btnSubmit, to the last row in the right column, with the Text, “Submit”. The form should now look similar to Figure 5.33.

Figure 5.33 – The ListBox and DropDownList Web and Html Controls



All that’s left to do is complete the implementation of the Button’s Click event handler. Double click on btnSubmit to create the event handler, and use the code in Listing 5.25.

Listing 5.25

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
```

```
   lblTitle.ForeColor = Color.FromName(lstFontColor.SelectedItem.Text)
   lblTitle.Font.Size = FontUnit.Parse(lstFontSize.SelectedItem.Text)
   lblTitle.BackColor =
 Color.FromName(lstBackgroundColor.Items.Item(lstBackgroundColor.Sele
 ctedIndex).Text)
   lblTitle.Font.Name =
 lstFontName.Items.Item(lstFontName.SelectedIndex).Text
 End Sub
```

Notice that even for the Html Controls, one line is all that is needed to obtain the required details about the selected item in each case. The first property to be modified is “ForeColor”. As was mentioned in the section detailing the Label control, this property modifies the font color. However, a System.Drawing.Color structure must be assigned to it – not a string. However, the Color structure includes a shared (i.e. non-instance) method, FromName that accepts a string as a parameter and returns the corresponding Color structure for the color name that was passed. To obtain the name of the color that was selected, the lstFontColor.SelectedItem property’s Text member was used, which returns the text property of the selected ListItem.

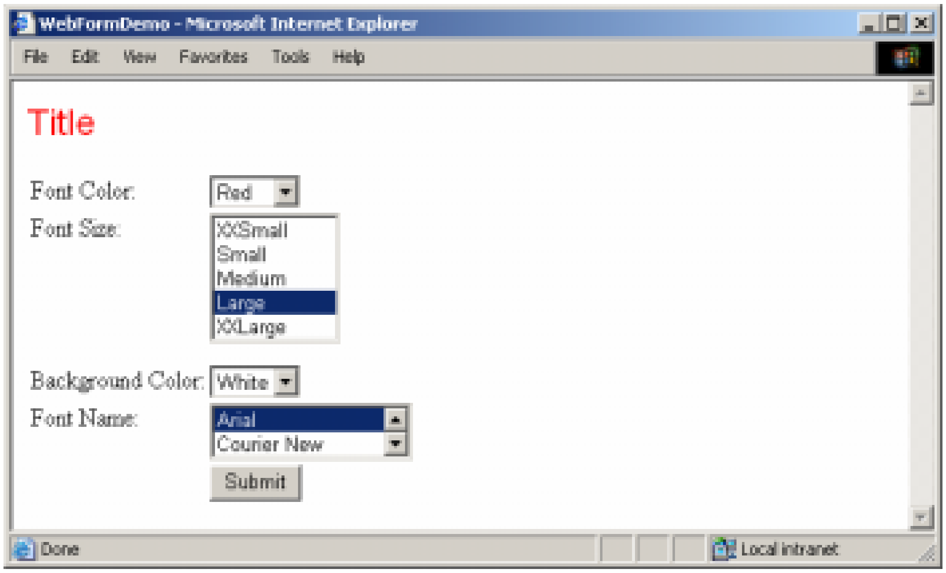
The Font.Size property, which was also covered in the earlier section dealing with the Label control, accepts a FontUnit structure, and again a shared method of this structure is used (Parse) to convert the string input into the acceptable structure. The syntax to obtain the text from the selected item is the same as for the DropDownList.

The BackColor property, which modifies the background color of the Label, also accepts a Color structure so again the Color.FromName method is used. However, the Dropdown Html Control needs a little more effort to obtain the text of the selected option. First of all, the lstBackgroundColor.SelectedIndex property provides an Integer representing the index of the selected item. The lstBackgroundColor.Items property returns a collection, and the Item index property of that is used to obtain a specific item from the collection by specifying the item’s index. Therefore lstBackgroundColor.Items.Item(lstBackgroundColor.SelectedIndex) returns the selected ListItem object for manipulation, from which point the Text property is used to obtain the text of the item.

Finally, the Font.Name Label property accepts a string of the font name to use, so there’s no need to modify the Text property value of the selected item in the Html Control Listbox – it can simply be assigned to the Font.Name property. Since the Html Control Listbox and Dropdown are actually both System.Web.UI.HtmlControls.HtmlSelect objects, the method to obtain the selected item’s text is obviously the same.

When the page is loaded, you can modify the properties of the Label by selecting various combinations of options in the list controls and clicking the “Submit” Button. Figure 5.34 shows the page in action.

Figure 5.34 - The ListBox and DropDownList Web and Html Controls in action



Title

Font Color: Red

Font Size: XXSmall, Small, Medium, Large, XXLarge

Background Color: White

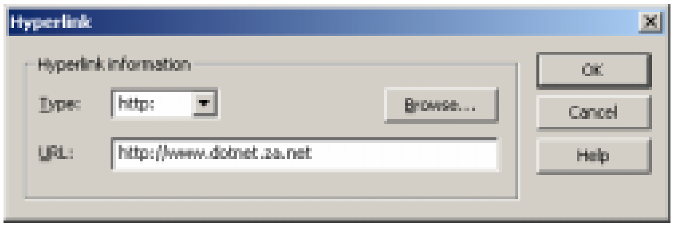
Font Name: Arial, Courier New

Should you wish to add or remove items programmatically, you can do so using the Items.Add and Items.Remove methods in exactly the same was as with the RadioButtonList. This applies to both the Web and Html Controls, even though their design-time behavior is significantly different.

The HyperLink

Much of the basis behind the World Wide Web relies on the concept of HyperLinks that connect related pages of information. The Visual Studio.NET IDE includes a menu option that will convert the selected text in the Designer into a hyperlink. To do this, simply type some text into the Designer, select the part you want to be a link, and click the Insert menu and choose the Hyperlink option. Figure 5.35 shows the dialog with a URL for the hyperlink inserted.

Figure 5.35 – The Hyperlink dialog



Hyperlink information

Type: http:

URL: http://www.dotnet.za.net

However, just like it is possible to add text to the page by simply typing in the Designer and yet there is still a Label control, even though you can add links without the aid of a server control, a HyperLink Web Control is still available. This provides the advantage of being able to programmatically modify where the link points to at run-time, and can play a very important role in state maintenance using querystrings, as will be discussed further in the next chapter.

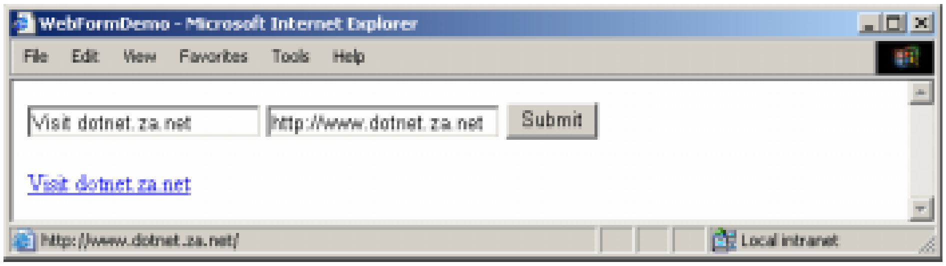
To demonstrate the HyperLink Web Control, we're going to build a page where both the text of the HyperLink and the URL that the control is pointing to will be changed via two TextBox Web Controls. To do this, first add two TextBox controls to the page and set their IDs to txtText and txtURL. Next add a Button with its ID set to btnSubmit, and finally add the HyperLink control at the bottom and set its ID to lnkDynamicLink. Double click btnSubmit to create a Click event handler, and use the code shown in listing 5.26.

Listing 5.26

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
  lnkDynamicLink.Text = txtText.Text
  lnkDynamicLink.NavigateUrl = txtURL.Text
End Sub
```

The Text property of the HyperLink sets (or returns) the text that is displayed on the page by it, and the NavigateUrl property is the URL that the anchor points to, and where browser will navigate to when the link is clicked. Figure 5.36 shows the HyperLink after its Text and NavigateUrl properties have been set programmatically.

Figure 5.36 – The HyperLink Web Control



Visit dotnet.za.net

http://www.dotnet.za.net

Visit dotnet.za.net

http://www.dotnet.za.net/

The HyperLink will typically not be used in this fashion, and for the most part, its properties will be set "behind the scenes" to help store application state by appending querystrings and so forth, but the basic concept of a programmable link persists and can prove to be very useful.

Conclusion

The purpose of this chapter has been two-fold. First and foremost, it has been to give you a solid introduction to the basic controls available for building your user interfaces, and to show you how to use these controls in both the design-time environment, and during run-time. If you are comfortable with using the controls in the Designer, I also encourage you to learn, or at least become familiar with, the notation and syntax that ASP.NET actually uses for controls in the UI HTML code, as although the IDE abstracts you from it, most documentation will provide ASP.NET examples in code, rather than Visual Studio.NET instructions.

However, there was in fact a second, more subtle purpose for this chapter, which was to help you become more familiar with the Visual Basic.NET syntax. Hopefully through the examples in this chapter, you will now be comfortable with these tasks, amongst others: declaring variables and creating objects in VB.NET; using the If conditional statement; using the For loop structure; accessing members of objects and classes; performing type conversions; assigning values to variables and properties, and other simple yet important tasks. You should also hopefully have a basic understanding of the event model in ASP.NET, and its place in the Page Cycle.

Chapter 6

Moving between Web Forms and Persisting State

Just as in traditional Windows application development, not all the operations of an application take place within just one form. In a typical Windows application there are many forms. These may include properties, file handling, printing and application settings dialogs. Likewise, a web application is made up of many pages, with each page serving a specific purpose, with specific tasks that it must perform. For example, an e-commerce book-selling site may have a page to browse through all the available titles, another page to view details on a specific title that the user selects, and yet another to place an order. This presents two problems to the developer – firstly, how to send the user from one form to another, and secondly, how to “remember” the options that they’ve chosen, for use by the page that they’re moving to, otherwise known as “persisting state”. There are numerous ways to tackle both problems, and each solution has its own advantages and disadvantages in different situations. This chapter shows you all the possible solutions, and gives example scenarios in which each solution would be most effective.

Adding Web Forms

Naturally before you can move between web forms, you actually need to have another web form to move to. To add another Web Form to your project, right-click on the project node in the Solution Explorer and choose “Add-Add Web Form”, as shown in figure 6.11. Enter a suitable name for the form and click “Open” in the dialog that appears. The Web Form will have been added and the Solution Explorer should have updated to reflect the change.

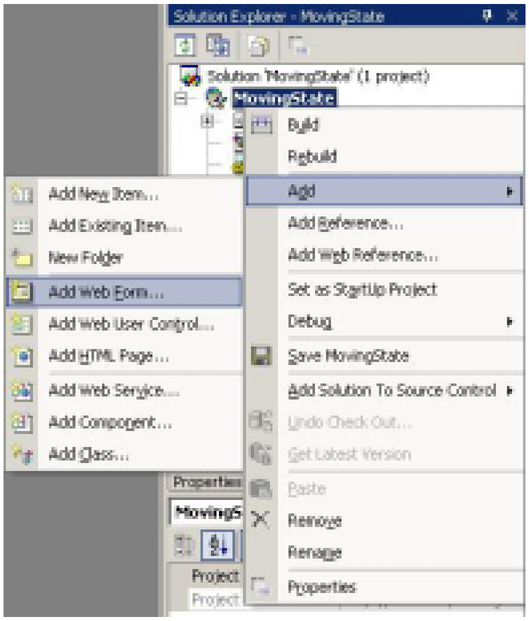


Figure 6.1 – Adding a new Web Form

Moving Between Web Forms

There are two basic categories of moving between ASP.NET pages – there is moving caused by an action by the user (such as clicking a link) and there is moving caused programmatically by the server (such as when you login to Hotmail, the login script checks your credentials, then automatically sends you to your inbox, or welcome page). In both categories there are some methods that are fairly obvious, and some that are slightly subtler, but this part of the chapter will deal with them all.

Client side-Induced Movement

There are four main ways in which the client (the browser and/or the user) can cause movement between pages in the site. These include links, HTTP redirection, forms and client-side script.

Anchors (Links)

The first and most obvious way for a user to cause them selves to be navigated from one page to another is by using a link. The previous chapter dealt with the HyperLink web control, but as a refresher, this control is basically an anchor tag (A) that is programmatically accessible on the server side. The use of this as opposed to a ‘static’ link (which will be covered shortly) will be made visible when state persistence is covered in the 2nd part of the chapter.

As an example, the following web control definition will result in a link being created to a page called “WebForm2.aspx”.

```
<asp:HyperLink id="lnkNextPage" runat="server"
NavigateUrl="WebForm2.aspx">Next Page</asp:HyperLink>
```

ASP.NET generates the following HTML when the page is loaded:

```
<a id="lnkNextPage" href="WebForm2.aspx">Next Page</a>
```

However, since ASP.NET processes the web control, it is possible to change the properties of the HyperLink control at “run time”, resulting in a different anchor tag being generated. For example, if in the Page Load event handler, the following line was included:

```
lnkNextPage.NavigateUrl = "WebForm3.aspx"
```

Then the link would point to WebForm3.aspx. This becomes particularly useful when transferring state from page to page.

Of course, if you’d like to include a link to another page and don’t need to change it programmatically, then simply inserting an HTML anchor tag straight into your page will suffice. In the Design view, you can select either an image or text and click Insert-Hyperlink. In HTML view, you must manually enter the anchor link. Note that in both the design and HTML views, the Properties window can be used to modify the properties of the tag.

HTTP Redirection

This method of the user moving from one page to another doesn’t actually involve any action on their part, but it does occur due to the behaviour of the web browser that they’re using. HTTP Redirection is a fairly common method of refreshing a page automatically, or automatically sending the user to another page after a specified time. The most common example of HTTP Redirection is probably its use in pages that inform the visitor that “This site has moved. You will be sent to the new address in 5 seconds.” This effect is achieved using the META tag in HTML. You will probably have noticed that Visual Studio.NET automatically inserts four META tags into all Web Forms. These store information about the Web Form for use by Visual Studio.NET. The HTML standard provides META tags for the purpose of supplying additional information about an HTML, and a META tag normally takes the following format:

```
<meta name=”keyname” content=”content”>
```

META tags can be used to store any information. For example, a META tag could be used to store the author of a document. In this case, the META tag might look like this:

```
<meta name=”Author” content=”Peter McMahon”>
```

Search engines use META tags to help better index pages in their engines. More specifically, they look for the Keywords and Description META tags and use these values.

However, HTTP Redirection uses a slightly different form of the META tag, and doesn’t include the name attribute. Instead the attribute “http-equiv” is supplied, with the value “Refresh”. The content attribute then stores the amount of time in seconds before the refresh will occur, and if applicable, the document that must be called, separated by a semi-colon and “url=”. If you wanted to automatically redirect a user from WebForm1.aspx to WebForm2.aspx after they’ve viewed WebForm1.aspx for 10 seconds, you’d insert the following META tag into the HEAD section of WebForm1.aspx’s HTML code:

```
<meta http-equiv=”Refresh” content=”10;url=WebForm2.aspx”>
```

After 10 seconds, the browser will then request WebForm2.aspx, as if it had been a link that the user had clicked on.

HTML Forms

ASP.NET goes a long way towards changing the idea of HTML forms as they have been traditionally known, and making HTML forms operate more like traditional Windows forms do, but HTML forms are still a very common and powerful mechanism for moving between web forms and particularly in persisting state. In chapter 3, it was explained that HTML forms contain an “action” attribute that allows the file where the contents of the form will be sent. For example, a form on one page could send its contents to another page that then processes the form. In ASP.NET the most common occurrence is the form data being sent back to the same Web Form that processes it before it displays the form again. Although for many developers this was common practice in ASP, many did not use it, but ASP.NET strongly encourages this style of development and makes it easier by providing a programming model for the

Web Controls, rather than having to constantly deal with Request.Form variables. This makes Web Forms development become significantly more like Windows development where forms are really self-contained units, but sometimes it is useful to revert back to the “traditional” method of sending form data to another file.

You come across forms that “link” to other pages every day. Possibly the most common example are search engines. When you enter a search term in Google and click Search, the data that you entered is sent to a different page. When you search for a book in Amazon.com using the search bar on the main page, the data isn’t sent back to the main page to be processed, but rather to their search script.

When using this method, there are several important changes in the code that need to occur for it to work. Firstly, the form must be a pure HTML form and not a server control, so the form must not have a runat=”server” attribute. This is because, when using server controls, ASP.NET forces the form to send all the data back to the same page. In other words, it forces the action attribute of the form to revert to the name of the current page, regardless of whether it is manually set differently or not.

Because the form is no longer a server control, and for a variety of other reasons, the page that the form data is being sent to will not be able to use the “friendly” method of obtaining form values, and will have to revert to the traditional ASP method of using Request.Form() that was briefly covered in the previous chapter.

In Listings 6.1 and 6.2, an extract from two web forms is shown. In Listing 6.1, a pure HTML form that will send its data to another page is shown, and Listing 6.2 shows the Page_Load event handler for WebForm2.aspx that assigns a value passed from the 1st page to a label control in WebForm2.aspx.

Listing 6.1

```
<form action="WebForm2.aspx" method="post">
 Enter your name.<br>
 <input type="text" name="txtName">
 <input type="submit" value="Send">
</form>
```

Listing 6.2

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
 lblName.Text = "Your name is " & Request.Form("txtName") & "."
End Sub
```

When the submit button on the 1st page is clicked, the browser submits a request to the server for WebForm2.aspx page, and includes the data from the form in the request. When WebForm2.aspx receives the request, it accesses the posted data using the Request.Form() property and assigns it to a label.

Client-side Script

Although not commonly used, there are several ways in which client-side script can be used to move the user between pages. Scripting can be used on its own to redirect the user, or it can be used in conjunction with regular HTML forms by submitting the form, even though the user has not clicked the “submit” button.

Firstly, the Internet Explorer DOM (document object model) exposes a location object that provides details about the current document’s location. However, assigning a value to the href property of the location object will result in a redirect to that document. The following code, if inserted into the Web Form’s HTML, will automatically redirect the user to WebForm2.aspx once the page has been loaded.

```
<script language="javascript">
 location.href = "WebForm2.aspx";
</script>
```

A history object is also exposed and includes a back() method. This method allows you to specify how many items back in the history the browser must move to. For example, inserting the following into a Web Form’s HTML will result in the user being automatically sent back to where they’d come from.

```
<script language="javascript">
 history.back(1);
</script>
```

However, it is important to note that, where possible, the browser will always attempt to load the item from its cache when this method is used.

To automatically submit forms, the form must have a unique name, which is then referenced using the document object, and is submitted using the submit() method. The following is an example of a form that is automatically submitted as soon as the page is loaded, and thus the user sent to WebForm2.aspx, along with the data in the form:

```
<form name=”Form1” action=”WebForm2.aspx” method=”post”>
 <!-- some form fields -->
 <input type=”submit” value=”Submit”>
</form>
<script language=”javascript”>
 document.Form1.submit();
</script>
```

Of course, these methods wouldn’t normally be simply placed in a script block, but rather included in an event handler. The major advantage that this offers is that you can perform certain operations before the user is sent to another page. ASP.NET’s validation controls, which are covered in the next chapter, use this technique. Listing 6.3 shows how to bring up a message box, then redirect the user when the mouse cursor moves over a link. This is not a practical example, but it demonstrates the concept.

```
<script language="javascript">
 function MyEventHandler() {
  window.alert(document.Form1.txtName.value);
```

```
    location.href = "WebForm2.aspx";
   }
  </script>
  <form name="Form1" method="post">
   Enter your name:
   <br>
   <input type="text" name="txtName">
   <a href="#" onmouseover="MyEventHandler();">Move your mouse
  here!</a>
  </form>
```

[Listing 6.3]

Server side-Induced Movement

ASP.NET provides several methods for redirecting users from server-side code. Each has its own specific purpose and accompanying nuances, which must be remembered when deciding which one is appropriate for a particular situation.

Response.Redirect

The Response.Redirect method has two overloaded versions. The most commonly used version takes one string parameter, which is the relative or absolute URL to the resource that the client must be redirected to. In effect, the first overloaded version of Response.Redirect terminates the processing of the ASP.NET page and the user is sent to another page. However, the second overloaded version takes the URL string string parameter, followed by a Boolean specifying whether the current response should be immediately terminated or not.

The one drawback of Response.Redirect is that in both cases it requires a roundtrip to the client before the redirection occurs, which can have an impact on performance, although generally this does not result in a major performance bottleneck.

Listing 6.4 shows an event handler for a button that will redirect the user when the button is clicked.

Listing 6.4

```
  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
  System.EventArgs) Handles Button1.Click
   Response.Redirect("WebForm2.aspx")
  End Sub
```

For this particular example, the exact same effect could have been gained by using the aforementioned HTML forms or client-side script techniques, but since the Redirect() method is executed in server-side code, it has decided advantages over the client-side methods. Possibly one of the most common occurrences of server-side redirects on the Internet are login scripts. When you log in to a web application, the site will normally send you from the form page to another page that checks your credentials. If you are successfully authenticated, you will then normally be redirected to a menu, or back to the main page. That same result would be difficult to obtain without using server-side redirection, and would be very cumbersome. This is just one in the many examples where server-side redirection is very useful.

By default, the endResponse parameter of Redirect() is True, and thus when calling the first overloaded version of Redirect() that only allows one parameter, the URL, the execution of the current ASP.NET page is immediately terminated. However, this is sometimes inappropriate as it may be convenient for the rest of the code to complete execution before the redirection occurs. Examples of such scenarios would include if a file has been opened, the script should complete so that it is closed again etc. In such cases, the second overloaded version of Redirect() should be used, with the second parameter set to False, as demonstrated in Listing 6.5.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
 'Code that opens a file
 Response.Redirect("WebForm2.aspx", False)
 'Code that closes the file
 'This code will be executed
End Sub
```

[Listing 6.5]

Server.Transfer

Microsoft introduced Server.Transfer in ASP 3 to alleviate the problem of the client roundtrip in Response.Redirect. However, in avoiding this problem, it creates another, which will be discussed shortly.

Server.Transfer has a very similar effect to that of Response.Redirect, but although the differences between the two are subtle, they must be taken into account. Firstly, the way in which Server.Transfer operates is significantly different. While Response.Redirect results in a command being issued to the client for a redirection, Server.Transfer results in ASP.NET immediately terminating to current ASP.NET page execution and beginning that of another, with the results of that page being sent to the client. Notice that there is no client roundtrip this time, but this, although theoretically increasing performance, does have one adverse affect. Since the client is not “informed” that a redirection is taking place, it assumes that the result that it is receiving is that of the page that it initially requested. This effectively results in the client incorrectly assuming the URL of the page that it is actually receiving, and this can cause problems with bookmarking.

To better illustrate this, take a login on a website as an example. Login.aspx contains a server control form (i.e. the page will ‘post back’ to itself) and the logic for authenticating users. If the user enters invalid credentials, then a message is displayed informing them of the situation, but if they are successfully authenticated, then they are redirected to a menu, for example, menu.aspx. However, when a user logs in and the menu is displayed, because the client browser isn’t aware of the redirection, the address bar in the browser will still indicate that the user is viewing login.aspx, even though they are actually viewing menu.aspx.

The Transfer() method too has two overloaded versions. However, unlike Redirect() it does not provide a parameter to specify whether the response must be ended immediately or not – it will always be terminated as soon as the method is called. Rather, the second overloaded version allows two parameters to be passed, the URL, and then a Boolean specifying whether or not the Form() and QueryString() collections from the current page should be automatically transferred to the page that is being redirected to. The advantage that this offers will become significantly clearer in the latter section of the chapter dealing with persisting state. Listing 6.6 demonstrates the use of the first overloaded version of Server.Transfer, including only one parameter.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
 Server.Transfer("WebForm2.aspx")
 'Any code from this line on will not be executed
End Sub
```

[Listing 6.6]

Server.Execute

Server.Execute is a hybrid between user controls and page redirection. User controls are covered later in the book, but in their simplest form, involve putting together a group of HTML elements or server controls that can be easily inserted into a page as a single entity. For ASP developers who haven’t come across Server.Execute previously, it can perform a similar function to server-side includes in ASP. Server.Execute is similar to Server.Transfer, in that it calls a specified page and transfers control over to the page, but the difference between the two methods is that when using Execute(), once the second page has completed its response, control is returned to the original page, where the code execution starts again where it left off.

Server.Execute can have several useful applications, but generally it is difficult to implement and the desired solution could probably be obtained much more easily using user controls, classes and procedural programming in ASP.NET.

The Server.Execute method has two overloaded versions – the first taking only a string parameter containing a URL, and the second both a URL parameter and an instance of the TextWriter class. The first version dumps the response of the URL immediately where the method is executed, but the second version places the response into a TextWriter object, allowing the response to be placed into a label control, or the like. Listing 6.7 demonstrates the use of this second version.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
 Dim objWriter As New System.IO.StringWriter()
 Server.Execute("WebForm2.aspx", objWriter)
 Label1.Text = objWriter.ToString()
End Sub
```

[Listing 6.7]

When using Server.Execute() in the fashion described above, it is important to note that the page will return the same response as if it was being requested by a browser. It is therefore important to build the pages that are being called using Server.Execute() accordingly by only including necessary code, and not including the HTML document “template” of HTML, META, BODY and FORM tags, in much the same way as User Controls are built.

Persisting State

It’s all fair and well moving between web forms, but there’s not much point if the application doesn’t know what has been done on the previous forms. An example of state information is a user ID. When you log in to Hotmail, your user ID has to be temporarily stored so that the application knows which inbox is yours, what your settings are for sending mail, what your signature is and a whole host of other items. In Windows application development, persisting state information is effectively automatic because the user is constantly ‘connected’ to the application, and so the application can easily check what options were selected on a form that was opened 10 minutes ago, or what your name is for appending to a document that you’re writing. However, the Internet is a stateless environment. This stems primarily from the fact that the user is not constantly connected to the application and a whole process of an application is not spawned when a user ‘opens’ the application. When a user requests an ASP.NET page, the browser opens a connection to the web server and requests the appropriate file. The server then processes the request, executes the code in the ASP.NET page and returns the result to the client browser, and the connection is then closed. A ‘copy’ of the application is not left running in the server’s memory, ready to resume activity when the user connects again to request another page. The primary reason for this is that it would simply be far too resource-intensive for web servers to implement. This means that the programmer has to provide some other method for the application to ‘remember’ whom the user was, and what settings they’d applied.

Multiple Solutions – Standard and Proprietary

The HTTP protocol and HTML standard doesn’t totally ignore state, and do provide several methods of maintaining state, each with its own advantages and disadvantages. These include query strings, hidden form variables and cookies. Microsoft, in addition to providing the ability to use the standard methods of state maintenance, also provides several other complementary, or independent methods, applicable to ASP.NET only, and not HTTP applications in general (although some other server-side programming frameworks, such as PHP, also offer similar functionality). These include session variables and application variables.

Persisting State using QueryStrings

The query string is the ampersand-separated name and value pair collection denoted by a question mark following the URL in an HTTP request. Take for example the following request:

```
http://localhost/default.aspx?myvariable=myvalue
```

The file that is being requested is default.aspx, and the query string contains one variable, “myvariable”, that contains the value “myvalue”. However, query strings can be used to pass numerous variables in the request by separating name and value pairs using an ampersand, as follows:

```
http://localhost/default.aspx?myvariable=myvalue&anothervariable=another
value&athirdvariable=athirdvalue
```

As the previous chapter demonstrated, it is possible to force forms to pass data using this format, rather than the ‘hidden’ method of HTTP POSTs by changing the form’s method attribute to “GET”. However, the real value of query strings comes because they can be passed using any request, not only using forms. For example, it is possible to create an anchor tag as follows:

```
<a href=”default.aspx?myvariable=myvalue”>Home Page</a>
```

Again, as was demonstrated in the previous chapter, it is possible to retrieve this value using ASP.NET server-side code using the Request.QueryString() property. In default.aspx, it would be possible to retrieve the value of the querystring variable “myvariable” as follows:

```
Dim strMyVariable As String = Request.QueryString(“myvariable”)
```

In the above case, the value of strMyVariable would be “myvalue”, but if no querystring was passed, an exception would not be raised, but the string would simply be empty.

In itself, defining anchors with query strings is not hugely useful in the context of maintaining state, because the values passed are still the same for every user. However, using server-side script to generate the link would be very useful, as the link could then be tailored for each user. There are several methods by which this can be done, the most obvious being using an HtmlAnchor control. To demonstrate this, place an HtmlAnchor control on a form, along with a TextBox and a Button. In the Page_Load event, write the following code:

```
HyperLink1.NavigateUrl = “WebForm2.aspx?name=” & TextBox1.Text
```

When the form is loaded for the first time, the link will point to “WebForm2.aspx?name=”. Type your name into the TextBox, click the Button and the link should point to “WebForm2.aspx?name=yourname”. When the user clicks the link, WebForm2.aspx now “remembers” what the user entered in the previous form, and can call upon it by referencing Request.QueryString(“name”).

Of course, this is a very simple example and the same result could have been achieved using other means, but this technique is very useful when the application needs to know values that were entered 2 or 3 forms back. However, as has been mentioned, using the HtmlAnchor class is only one of the ways in which the anchor tag can be dynamically generated.

Many Web Controls can contain other controls. Those that include a “Controls” property allow other controls to be programmatically added and included within them. Amongst several of these “Container” controls, is the Panel control. A dynamically built link could be added to one of these controls, either by adding a HyperLink control, or by adding a LiteralControl object. The use of the HyperLink control has already been discussed so will not be elaborated on, but the LiteralControl class provides an interesting method of adding items to a page that don’t need to be programmatically accessed, and operates in a very similar fashion to the Literal Web Control.

The LiteralControl class's constructor allows a string parameter to be passed, containing the HTML code to be inserted. Listing 6.8 shows its use, in conjunction with a Panel, as an alternative to using a HyperLink control in the same example as above.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
 Panel1.Controls.Add(New LiteralControl("<a
href=""WebForm2.aspx?name=" & TextBox1.Text & """>Next page</a>"))
End Sub
```

[Listing 6.8]

Take note of the use of the double-inverted commas, used as an escape character for inserted quotes, which would normally be a string delimiter. This code would have the exact same result as the previous example, namely generating a link that passes the text in TextBox1 as a query string variable.

Another method of generating the link and inserting it into the document is using a Literal Web Control, which was again covered in the previous chapter, and allow HTML code to be included in its Text property. It functions very similarly to the LiteralControl class, but is slightly different in that it is a Web Control and can therefore be added at design-time. After adding a Literal control, the following code could be used in the Page_Load event to create the link with a query string:

```
Literal1.Text = "<a href=""WebForm2.aspx?name=" & TextBox1.Text &
""">Next page</a>"
```

Any Web or Html control that has a Controls property, or has a Text property, can be used to dynamically insert anchors into a page using the previously demonstrated methods. There are however two more completely different methods that were very commonly used in previous versions of ASP, where page content was generally not separated from the server-side code by a code-behind, as the ASP.NET programming model and VS.NET encourage. This said, it is still possible to include server-side ASP.NET code in amongst the UI code using the traditional <% and %> delimiters. For example, inserting the following code in the UI file will result in a link being dynamically created, as with all the previous examples:

```
<%
 Response.Write("<a href=""WebForm2.aspx?name=" & TextBox1.Text &
""">Next page</a>")
%>
```

Where the link is created in the page depends on where this code is inserted. The overloaded version of Response.Write() in this instance takes one string parameter which it writes directly to the response being generated in the position where the code is currently executing.

Following on from the above “inline server-side code” method, there is another ASP.NET delimiter which allows server-side variables to be written to the response as if done using a Response.Write(). This delimiter is <%=, and ended by %>. The value of a variable name included between these two delimiters is written to the response. Therefore, the below code produces the same result as well:

```
<a href=”WebForm2.aspx?name=<%=TextBox1.Text%>”>Next page</a>
```

This code would be placed where the link must appear in the page, as is the case with the previous example of inline server-side code.

Persisting State using Hidden Form Variables

One of the most common methods of persisting state between different pages when movement is caused by submitting forms is the use of hidden form variables. In fact, ASP.NET itself uses hidden form variables to maintain state on controls on form postbacks. For an example where this method may be appropriate, consider filling out a registration that spans 3 or 4 pages. The application obviously needs to remember what the user filled out on the previous pages when the final page is reached so that the registration can be completed. Whether it remembers only an ID for the new user generated by the first page so that it knows what information the user submitted, or it transfers all previously entered information in separate hidden form fields, state information must be persisted, and for a forward-only movement in forms pages, using hidden form fields is ideal. Of course, it isn’t the only method to achieve the desired result though – a querystring on the URL in the form’s action attribute could be used for this purpose in the same manner as the previous section has discussed, or session variables could also be used, as will be discussed in the next section.

Dynamically generated hidden form fields can be generated using almost exactly the same methods as used for generating links in the previous section. Firstly, the Html Hidden Input control (HtmlInputHidden class) can be inserted and its Value property assigned in the Page_Load event. For the examples for the hidden form technique, assume that the page that is being demonstrated is the 2nd in a chain of pages containing forms that move from one to the other in a sequence (as in a registration). Assume that the first page contains numerous text fields, of which one is called txtName. For this example, only this value will be persisted to demonstrate the concept. The following code would persist the txtName value in the 2nd page using an Html Hidden Input control named hidName.

```
hidName.Value = Request.Form(“txtName”)
```

In following pages, this exact line could be repeated (assuming that the hidden field is also included), persisting this value from the first page, through all the pages inbetween, right through to the last.

Again, the hidden field could also be added as a LiteralControl in a Panel’s Controls property. This could be achieved using the following code in the Page_Load event, assuming that a Panel named Panel1 exists within the confines of the form (between the form start and end tags).

```
Panel1.Controls.Add(New LiteralControl("<input type=””hidden””
name=””txtName”” value=”””” & Request.Form(“txtName”) & “”>”))
```

Similarly, a Literal web control’s Text property could be set in Page_Load to achieve the same result:

```
Literal1.Text = "<input type=""hidden"" name=""txtName"" value="""" &
Request.Form("txtName") & "">"
```

Lastly, inline code could also be used to the same effect when placed in an appropriate location in the UI code (again, between the form’s opening and closing tags).

```
<%
 Response.Write("<input type=""hidden"" name=""txtName"" value=""" &
Request.Form("txtName") & """>")
%>
```

Finally, the required variable could be inserted into a static hidden field using the following code:

```
<input type="hidden" name="txtName"
value="<%=Request.Form("txtName")%>">
```

Persisting State using Session Variables

Session variables offer a method of persisting state that is entirely different from the first two methods presented. It offers a very useful facility that is available in any page in the application without including specific code to maintain it on every page, as was the case with the previous methods. This makes it very easy to use and very versatile, as it can be used on any page, regardless of whether a form was used, and without worrying whether a query string was accidentally removed.

As the “Session” property of the Page class, the HttpSessionState object associated with every page contains information unique to each user of the application. ASP.NET manages the session state in its entirety, and determining which user is requesting a page, deleting redundant content and the multitude of other administrative tasks are all performed automatically by ASP.NET, hidden from the developer. Simply put, the Session object provides a collection of name and value pairs that are unique to each user that are set programmatically in the server-side code, with all maintenance tasks performed behind-the-scenes by ASP.NET.

One of the major problems of using query strings and form variables to persist state is that users can easily modify them. For example, if a query string variable is being used to store the id of a user, the user can simply modify the id in the query string in the address bar of his or her browser, and the application can be fooled into believing that the user logged in as someone else, which obviously poses a huge security risk. However, Session variables are stored on the server only, and none of the session variable values are ever stored on the client, thereby alleviating this potential security risk.

The Session object (an instance of the HttpSessionState class, and member of the Page class) contains numerous methods and properties for inserting, viewing and deleting items from session state. The Add() method accepts two parameters – a string as a name for the variable to be added, and an object that will be the value of the variable.

> Whilst the Add() method can accept an object, it is advised that only values of basic data types, such as strings, integers and dates, be inserted due to considerable strain that may be put on the server, baring in mind that a new collection of session state variables is created for every user of the website, and this collection is only deleted after a relatively lengthy period of time.

The Remove() method accepts one string parameter containing the name of the variable to remove from the session variables collection.

To retrieve items from the Session object, the Item() property is used. The Item property accepts a string parameter of the name of the item, or an integer representing the index of the item, and returns the value or object. This value will be returned in the same type as it was inserted as. For example, if a string was added, when it is retrieved, a string variable will be returned without having to perform any casting. Item() is the default property of the Session object, and therefore it is also possible to reference session variables using Session(). The Item() property is read and write, so items can also be added with it.

To demonstrate the use of Session state, listing 6.9 shows the code behind for a page containing a TextBox and a Button. When the button is clicked, a Session variable is created containing the value in the TextBox. The user is then redirected to the page with the code behind shown in Listing 6.10, where the value is then placed into a label on the second page without passing any form and query string values.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
 Session.Add("name", TextBox1.Text)
 Response.Redirect("WebForm2.aspx")
End Sub
```

[Listing 6.9]

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
 Label1.Text = Session.Item("name")
End Sub
```

[Listing 6.10]

Take note that no variables were passed in the Response.Redirect() call. After trying out this example, you can go to other forms in the project, either through links, or by typing the direct URL in the address bar of the browser, but when you return to the 2nd web form through whatever means, the value of the label will still be loaded as the value you originally entered on the 1st web form (assuming that the session hasn’t timed out).

To demonstrate the different ways in which items can be added to session state, this line could be used to replace the Add() method call in listing 6.9:

```
Session.Item("name") = TextBox1.Text
```

Because Item() is the default property, the line could also be replaced with this:

```
Session("name") = TextBox1.Text
```

Likewise, in listing 6.10, the code inside the event handler could be replaced with this line:

```
Label1.Text = Session("name")
```

Using the overloaded Integer version of the Item property, this line could also be replaced with this:

```
Label1.Text = Session(0)
```

When used in conjunction with the Count property of the Session object, a list of all the set session variables can be accessed, as demonstrated by this code:

```
Dim i As Integer
Label1.Text = ""
For i = 0 To Session.Count - 1
 Label1.Text = Label1.Text & Session.Item(i) & "<br>"
Next
```

> This is a demonstration of the Count property, and looping through and displaying all the session variables in the Session object should generally not be done in applications, since data stored in session variables might not be displayable using a label without serialization, since it is not guaranteed that only strings, integers etc have been stored in the Session object.

Session.Item() has been introduced as a method of accessing or assigning values to Session variables. It is however important to note that when values are assigned to Session variables using this method, the item is not always being added – if it already exists, the item will be edited, and so in the following code the first line might create the variable and assign it a value, but the second line modifies its value:

```
Session.Item("MyVariable") = "Test"
Session.Item("MyVariable") = "Edited value."
```

Likewise, if the Add() method is called twice referencing the same variable, the value in question will be overwritten:

```
Session.Add("MyVariable", "A value")
Session.Add("MyVariable", "Edited value.")
```

Traditional ASP developers may be wondering what happened to Session.Contents. Contents was previously the property of the Session object that was used to access session variables (i.e. the ASP 3.0 equivalent of the ASP.NET Session object’s Item property). However, in ASP.NET, Microsoft has made migration as easy as possible, and to this end, the HttpSessionState class does include a Contents property. This property returns a reference to the Session object, and therefore it is possible to use the Session.Contents(“variablename”) syntax as in ASP 3.0, because Item() is the default property of the HttpSessionState class. For the sake of completeness, the following line could also be used instead of the code in the event handler in listing 9.10:

```
Label1.Text = Session.Contents("name")
```

Although in most scenarios querystrings and form variables can be used to persist state, session variables often provide a solution that is much easier to implement and also offers several security advantages by avoiding the problem of malformed querystrings and form variables, either through accidental corruption or modification by malicious users. The chapter “Session State Model” discussed ASP.NET session state is much greater detail, and provides information for advanced users, especially for developers of web applications deployed in a web farm environment.

Persisting State using Application Variables

Application variables are in many respects very similar to session variables. They are set and accessed almost identically, they are accessible throughout the application, irrespective of querystrings and form variables passed and are not susceptible to the same possible security vulnerabilities that using querystrings and form variables have, amongst other similarities. However, the fundamental difference between Application and Session variables is that Applications variables are accessible by all the users of the application, whereas Session variables are unique to each user. Application variables do not time out as session variables do, but their lifecycle begins when the application is first accessed by any user, and ends when the application is stopped, as opposed to session variables' lifecycles that begin when the user first hits a page in the application, and ends when the session for that particular user times out after a certain period of inactivity.

While session variables are primarily used to store information about a specific user, application variables are generally used to store information about the application that all users should be able to access. A session variable might store the name of a user, or include the contents of a user's selections for a shopping cart, but an application variable would include items that are applicable to all users, for example, the physical path of the application on the server, or the number of users currently viewing the site (i.e. the number of active sessions).

As has already been mentioned, setting and accessing application variables is almost identical to the equivalent operations with session variables. The Application property of the Page class is an instance of the HttpApplicationState class, which provides an Item() default property for accessing application variables. The Add(), Set() and Remove() methods are also provided for adding, updating and removing application variables respectively. Because the Application object is simultaneously available to all users of the application, the methods Lock() and Unlock() are provided to prevent users from modifying the object at the same time. The use of these two methods is the one pivotal difference between the manipulation of variables in the Application and Session objects, as will be demonstrated shortly.

The advantages of the Application object are very apparent when used in conjunction with the Global.asax file (which is covered in more detail in the chapter "ASP.NET Application Configuration"). Simply put, the Global.asax file provides a place to include event handlers that are fired when events dealing with the application as a whole and user sessions occur. A counter of the number of concurrent users visiting a site is quite a common occurrence (for example, "You are 1 out of 251 people currently viewing this website."), and one of the ways of building such a feature is by using Application variables, and the Global.asax file. To build an "active visitors" counter sample, perform the following steps:

1. Start a new web form project.
2. Open the Global.asax file's code-behind (Global.asax.vb) and insert the following code inside the class implementation:

```
Sub Session_OnStart(ByVal sender As Object, ByVal e As EventArgs)
Application.Lock()
Application.Item("CurrentUsers") =
CInt(Application.Item("CurrentUsers")) + 1
```

```
Application.UnLock()
End Sub

Sub Session_OnEnd(ByVal sender As Object, ByVal e As EventArgs)
Application.Lock()
Application.Item("CurrentUsers") =
CInt(Application.Item("CurrentUsers")) - 1
Application.UnLock()
End Sub
```

3. Open the web form and insert a label named lblCurrentUsers.
4. Write the Page_Load event handler as follows:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
lblCurrentUsers.Text = "There are " &
CStr(Application.Item("CurrentUsers")) & " users currently visiting the
site."
End Sub
```

5. Compile the project and open the web form. The label should report than there is 1 active visitor.
6. Open another instance of Internet Explorer, and navigate to the web form. The label should now read that there are 2 concurrent users (if it doesn’t, refresh the page).

Without focussing too heavily on the Global.asax file, the two event handlers inserted fire when a new user session is started, and when a session ends. An application variable, “CurrentUsers” is used to store an Integer value representing the number of current users. It does this by having one added to it every time a new session is created, and one subtracted every time a user session ends. In the Session_OnStart event handler, the Application object is locked, so that it can only be edited by the current instance to avoid synchronisation errors, and the value contained in the Application variable “CurrentUsers” is then incremented. The Application object is then unlocked, allowing the Application object to be written to by other instances again. Similarly in the Session_OnEnd event handler, the Application object is again locked and unlocked when modifying the object. However in this event handler, the “CurrentUsers” variable is decremented. In the web form, the Page_Load event handler loads the value from the “CurrentUsers” Application variable and assigns it to the label.

After being introduced to the Session object, the use of Application variables should look very familiar – excepting the locking and unlocking of the object before writing to it, they are accessed and modified in exactly the same fashion as Session variables. As with the Session object, the Application object too allows numerous ways to add, access, edit and delete its variables.

First and foremost is the use of the Item() property, which allows variables to be added, edited and accessed. Since Item() is the default property of the Application object, it would be possible to replace the code in, for example, the Session_OnStart event handler to this:

```
Application.Lock()
Application("CurrentUsers") = CInt(Application("CurrentUsers")) + 1
Application.UnLock()
```

Likewise, the web form’s Page_Load event could have accessed the variable without including the default property name:

```
lblCurrentUsers.Text = "There are " & CStr(Application("CurrentUsers")) &
" users currently visiting the site."
```

However, unlike with the Session object, the Add() method can only be used to add variables, but not edit them. However, the Set() method can be used for this purpose. The following code would create, and then modify the application variable “MyVariable”:

```
Application.Add("MyVariable", "Original value")
Application.Set("MyVariable", "New value")
```

Modifying variables using the Application.Item() property is another method of editing application variables. The following code would create, and then immediately edit an application variable called “MyVariable”:

```
Application.Add("MyVariable", "Original value")
Application.Item("MyVariable") = "New value"
```

Application state has traditionally been used primarily for caching, since an object such as a database table can be stored once and accessed by all users of the application. However, ASP.NET introduces the Cache object that should be used for this purpose since it provides numerous advantages over using the Application object.

Data caching is covered in the chapter “Caching”

As has been shown with this section’s session counter, there are also other uses for Application variables. A common use that still applies in ASP.NET is the use of Application variables to store configuration strings for use in the code – these might include items such as the connection string for a site’s database connection, or the physical file path of the site. Any variable that needs to be globally accessible to all users should be stored in an application variable.

Lastly, to demonstrate a fairly common practical use of application variables, a chatroom will be built. One of the most popular methods of building an online “chatroom” is using application variables. The method by which this works is that the last 20 messages are stored in application variables. When the page displaying the messages loads, it displays the 20 application variables. When a user sends a message, all the messages stored in application variables are moved down one, and the user’s message is placed in the first position.

The form structure is very simple – add two textboxes, a button and a Literal control, separated by line breaks. Name them txtName, txtMessage, btnSend and litMessages respectively. Listing 6.11 shows the code-behind event handlers, subs and functions that will complete the functionality for the web form.

```
Private Function GenerateMessageList() As String
 Dim strMessages As String
 Dim i As Integer
 For i = 1 To 20
  strMessages = strMessages + CStr(Application.Item("Message" + CStr(i)))
+ "<br>" + vbCrLf
 Next
 Return strMessages
End Function

Private Sub RefreshDisplay()
 litMessages.Text = GenerateMessageList()
End Sub

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
 RefreshDisplay()
End Sub

Private Sub btnSend_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSend.Click
 If txtMessage.Text.Length <> 0 Then
  Dim i As Integer
  Application.Lock()
  For i = 20 To 2 Step –1
   Application.Item("Message" + CStr(i)) = Application.Item("Message" +
CStr(i - 1))
  Next
  Application.Item("Message1") = Server.HtmlEncode("[" +
Now.ToString("t") + "] <" + txtName.Text + "> " + txtMessage.Text)
  Application.UnLock()
  txtMessage.Text = ""
 End If
 RefreshDisplay()
End Sub
```

[Listing 6.11]

When the page is loaded, the current messages are loaded into the Literal control. If no message is entered and the form is submitted, then the messages are simply updated, but if a message was entered, it is placed at the top of the list (application variable “Message1”). The GenerateMessageList() function returns a string containing all the messages by looping through the application variables from “Message1” to “Message20”. Similarly the button’s click event handler loops through the messages from 20 down to 2, replacing each with the next message, thus message 20 will contain message 19, and 19 will contain 18 etc. “Message1” is assigned as the message that the user entered, preceded by the time that the message was sent, and the name of the user.

As demonstrated, a simple chat application can be built very quickly and easily using application variables. When used with multiple users, this is a good example of how different users can edit application variables, and the changes are reflected for all the users of the application.

Persisting State using Cookies

Cookies, simply put, are small ASCII text files that reside on the user's machine. These files are created by the web browser on receiving the instruction from a website. A website can therefore "create" cookies, and then retrieve their value at a later stage. Cookie values can be modified by the website. An expiration date for a cookie is normally set to ensure that the cookie is deleted after a certain period of time, or by a certain date. For security reasons, only the site (or the "domain") that sets, or creates, the cookie is allowed to access it.

Cookies can be thought of as session variables that persist between sessions. Session variables exist only for the time that the user is accessing the website, and once the user leaves the website and the session times out, the session variables associated are deleted. However, cookies can exist and persist information for months because they are stored on the user's machine, thereby not using up resources on the server.

> Whilst it may sound that cookies are almost an extension of session variables, if anything, the converse is true – session variables actually rely on a cookie that is automatically sent with all ASP.NET responses to keep track of the user's session id so that the correct session variables can be matched with the correct user when session variables are set, edited and deleted.

One of the disadvantages of cookies (and therefore inherently Session variables) is that cookies rely on the browser to store the cookie. If the client browser does not support cookies, or the user has disabled cookies, then cookies will obviously not function. However, most users do have cookies enabled, and all of the current browsers support cookies.

Site personalisation is one of the foremost uses of cookies. Since cookies can be used to persist information for lengthy periods of time, they enable sites to "remember" users' identities when users return, after weeks, or even months. When you personalise a site, you will typically be asked your name and be presented with a few options of what information you'd like displayed when you visit the website. Generally the site will generate a unique ID for you and store your preferences in a database. Your ID will then be stored on your computer as a cookie, so when you return to the site, the cookie is requested and your ID obtained, and from there the site will load your preferences out of the database and use them to appropriately display the page that you requested. The site could of course store all the preferences as cookie values, which would alleviate the need for the database, but generally the site is easier to maintain when the preferences are stored on the server and only the user ID stored on the client.

Working with cookies is slightly different to dealing with session variables in ASP.NET code. To add a cookie, an instance of the HttpCookie class must be created. This represents a cookie. In its constructor the name, or the name and the original value, of the cookie can be specified. The properties of the HttpCookie object allow the expiry, domain and other settings of the cookie to be specified. In most cases the defaults will suffice, with only the name and value being separately specified. The following code will create a cookie called “MyCookie” and specify its value as “MyValue”:

```
Dim objCookie As New HttpCookie()
objCookie.Name = "MyCookie"
objCookie.Value = "MyValue"
```

The cookie could also be created using another overloaded version of the constructor, like this:

```
Dim objCookie As New HttpCookie("MyCookie", "MyValue")
```

As a slight deviation from session and application variables, the value entered for the cookie is a string, and can only be a string – not an Object, as is the case with session and application variables. This is because the value must be stored in an ASCII text file by the user’s browser, and must also be transported via HTTP, and neither have support for dealing with .NET objects.

After the cookie has been created, it must then be appended to the HTTP response. The Response object’s Cookies property is an HttpCookieCollection class, which includes an Add() method for adding cookie objects:

```
Response.Cookies.Add(objCookie)
```

If a cookie of the same name as one that already exists is added, the original cookie will be overwritten with the new one, as is the case with the Add() method when dealing with session variables. However, the HttpCookieCollection class provides the Set() method for the specific purpose of updating cookies. This code would update a cookie, “MyVariable”:

```
Dim objCookie As New HttpCookie("MyCookie", "New value")
Response.Cookies.Set(objCookie)
```

The HttpCookieCollection’s Item() property provides a reference to a cookie. The Item() property is overloaded, and the string name of the cookie, or a numerical index can be passed. The property returns an HttpCookie class representing the cookie requested. Therefore the following code could be used to modify the “MyCookie” cookie:

```
Response.Cookies.Item("MyCookie").Value = "Another new value"
```

Cookie Expiration Dates

Cookie expiry is set through the HttpCookie class’s Expires property, which is a DateTime value. An absolute expiration can be set (eg. All cookies expire at midnight on 2003/10/01), or a sliding scale can be used where the cookie expires and is removed 10 minutes from when it is created. The following code creates a cookie that will expire on the 1st October 2003:

```
Dim objCookie As New HttpCookie()
objCookie.Name = "MyExpiringCookie"
objCookie.Value = "MyValue"
objCookie.Expires = DateTime.Parse("10/01/2003 12:00:00")
```

A sliding scale can be created using a TimeSpan class, or with only the DateTime class. The following code creates a cookie that will expire in 10 minutes from when it is created, using the DateTime class only:

```
Dim objCookie As New HttpCookie()
objCookie.Name = "MyExpiringCookie"
objCookie.Value = "MyValue"
objCookie.Expires = DateTime.Now.AddMinutes(10)
```

However, the TimeSpan class can also be used. The following code creates a cookie that will expire 1 hour, 10 minutes and 5 seconds from when it is created:

```
Dim objCookie As New HttpCookie()
objCookie.Name = "MyExpiringCookie"
objCookie.Value = "MyValue"
Dim objTS As New TimeSpan(0, 1, 10, 5)
objCookie.Expires = DateTime.Now.Add(objTS)
```

After the expiry period has elapsed, the cookie will no longer be accessible and will have to be recreated.

Sub-values

It is possible for a single cookie to contain numerous key and value pairs. The Values property of the HttpCookie class is a NameValueCollection, which allows key and value pairs to be added to the cookie, in addition to the cookie's Value property value. The NameValueCollection class provides an Add() method for adding keys, and their respective values, and an Items() property for accessing and editing keys. This allows a cookie to contain not one, but numerous values. This is particularly useful if a group of settings are being stored using cookies, and they are all related. For example, a cookie storing information about the user could contain key and value pairs storing his/her name, shipping address and telephone number, all in one cookie.

The following code creates a cookie that includes several key and value pairs:

```
Dim objCookie As New HttpCookie()
objCookie.Name = "UserInfo"
objCookie.Value = "UserInfo"
objCookie.Values.Add("Name", "Peter McMahon")
objCookie.Values.Add("Telephone", "1234567")
objCookie.Values.Add("Address1", "123 Some Road")
objCookie.Values.Add("Address2", "Somecity")
objCookie.Values.Add("Address3", "123ZIP, SOMESTATE")
```

The Set() method can be used to modify values, as can the Item() property:

```
objCookie.Values.Set("Name", "P McMahon")
objCookie.Values.Item("Telephone") = "01234567"
```

The Item() property can also be used to access the value of a key/value pair:

```
Dim strName As String = objCookie.Values.Item("Name")
```

Retrieving Cookie Values

After a cookie has been set, unless it has expired or been manually removed by the user, it can be accessed at any time, from any page in the website that created it. The Request object’s Cookies property is again an instance of the HttpCookieCollection class, and it is from here that the relevant values can be retrieved. The Item() property can be used in the same way as with the Response.Cookies object, with the parameter specifying either the name of the cookie, or its numerical index. An instance of the HttpCookie class will be returned, and from there, either the Value property can be used, or if applicable, the Values property can be used to obtain the key/value pairs stored within the cookie.

To retrieve the value from a cookie “MyCookie” and store it to a variable, the following code would be used:

```
Dim strMyCookie As String = Request.Cookies.Item("MyCookie").Value
```

Similarly, the values of several key/value pairs from a cookie “UserInfo” could be extracted and stored in variables as follows:

```
Dim strName, strAddress1, strAddress2, strAddress3 As String
strName = Request.Cookies.Item("UserInfo").Values.Item("Name")
strAddress1 = Request.Cookies.Item("UserInfo").Values.Item("Address1")
strAddress2 = Request.Cookies.Item("UserInfo").Values.Item("Address2")
strAddress3 = Request.Cookies.Item("UserInfo").Values.Item("Address3")
```

Conclusion

Moving between web forms is an important aspect of any web application, but since HTTP and the Web are inherently stateless, user and application state must be manually maintained. Fortunately ASP.NET provides numerous methods of maintaining state that make the job significantly easier. The chapter also dealt with how state can be maintained using only intrinsic capabilities of HTML and HTTP, such as querystrings and hidden form variables, along with how ASP.NET makes using the HTTP state mechanism, cookies, easier to allow state to be persisted over long periods of time. The session and application state functionality of ASP.NET was also introduced.

Chapter 7

User Input Validation

As ironic as it may seem, users have been the bane of all programmers' existences since the very beginning of the profession. Why? Simply because users never enter the data they're supposed to, or in the way that it's supposed to be entered. Of course, no amount of coding can substitute a simply, logical interface that acts as a user would expect it to, but however well-designed an application is, there will always be users who entered data incorrectly, either because they don't read instructions or follow hints, or simply because they make mistakes. Whatever the cause, when an application receives data that it isn't expecting, it must know how to handle it. It can't simply do nothing, the application mustn't crash, and it mustn't deliver cryptic error messages, such as "ASP.NET error on line 21" that don't mean anything to the user, and least of all guide them to whether they made their mistake so that they can correct it. The application must handle data that isn't entered, or entered incorrectly, guide the user to where they made their mistake and tell him/her what they must do to correct the problem.

User Input Validation deals with validating data entered by users against a certain set of rules. In traditional ASP, all validation had to be done manually. The programmer had to write his/her own functions for checking that required fields weren't omitted, or ensuring that telephone numbers were entered in the correct form and that the starting dates entered were less than the ending date. However, ASP.NET provides several server controls that make validating user input very simple indeed, and in most cases, not a single line of code is required. These controls make what was once an arduous and extremely boring task very quick and easy. The collective functionality offered by the controls includes ensuring that required fields are filled in, comparing fields to constants or other fields, ensuring that fields' values fall within a specific range, making a field conform to a regular expression, along with a customisable control that provides a framework for custom validation code.

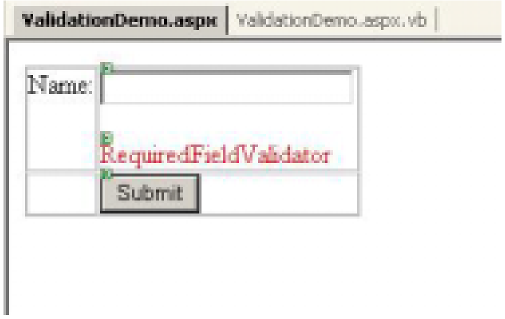
Like many of the advanced server controls, the validation controls make use of uplevel browser features when they are available, but they still function correctly when used in downlevel browsers. I.e. when DHTML support in available on the client, then client-side scripting is used to perform the validations, thus saving roundtrips to the server, but server-side code is always executed to ensure that validation rules are adhered to. In this manner users of uplevel browsers have the convenience of validation taking place on the client-side, alleviating the necessity of a roundtrip to the server and waiting only to be told that they left out a required field, and yet the security of the validation cannot be compromised because it is always run on the server-side, regardless of the client browser.

RequiredFieldValidator

Possibly the most common use for validation is to ensure that all required fields on a form are filled out. For example, in a registration form, such fields would include name, e-mail address and telephone number. Ensuring that such fields are filled out is the work of the RequiredFieldValidator control. Each field that is required must have its own RequiredFieldValidator control, which has a property that defines which control it is to validate. As with all the validation controls, a message must be supplied that is displayed when the validation fails – i.e. the user does not fill out a value for the required field.

To try out the RequiredFieldValidator, perform the following steps:

1. Create a new Web Form
2. Add a label, textbox, button and RequiredFieldValidator control as shown in figure 7.1.



3. Name the controls lblName, txtName, btnSubmit and valName respectively.
4. Set the ErrorMessage property of valName to “Name is a required field.”
5. Set the ControlToValidate property of valName to “txtName”.
6. Compile the project and load the Web Form. Initially only the label, textbox and button will be shown.
7. Try clicking the button without filling out a value. The message “Name is a required field.” should appear, and you will notice that the form did not post back.
8. Fill out a value in the textbox and retry submitting the form. This time the form will be submitted.

Not a single line of code was required, and the RequiredFieldValidator wrote both client- and server-side validation code to ensure that the txtName field was filled out. The ErrorMessage property of the RequiredFieldValidator control was used to specify what message must be displayed when validation fails, and the ControlToValidate property specified what control must be validated.

Because you viewed the page using IE, only the client-side script validation was demonstrated. To prove that server-side validation code was indeed generated, turn off client-side scripting in Internet Explorer by performing the following steps:

1. Open “Internet Properties” from the Control Panel
2. Switch to the “Security” tab
3. Select “Local Intranet”
4. Click “Custom Level”
5. Under the “Active Scripting” node, which is a child of the “Scripting” node, ensure that the “Disable” radio button is checked.

6. Click “Ok” and confirm that you do want to change the settings in the message box that appears.
7. Click “Ok” again to close the “Internet Properties” window.

Now open Internet Explorer again and navigate to the web form (or compile and debug the application in VS.NET). If you submit the form without filling in the txtName field, the form will be submitted and a postback will occur, however, when the page is returned, the validation error message is now displayed. If a value is filled in for the txtName textbox and the form is re-submitted, the message will have been removed. There is therefore no way to “fool” the ASP.NET validation controls into not validating the data because it knows the browser supports scripting (it’s IE), and yet the scripts are not executed because they’ve been disabled. In essence the server-side code is the important validation code, and the client-side code is simply provided as a convenience to be used when possible.

RequiredFieldValidator Options

The RequiredFieldValidator control features numerous properties that allow the control to be customised for most needs. Since the display mechanism for showing the error message when validation fails is similar to that of a Label control, most of the properties of a Label are present, allowing background and foreground colours, along with font styles and sizes to be changed. However, the RequiredFieldValidator also includes two properties that are unique to the validation controls. These are Display and EnableClientScript.

EnableClientScript Property

EnableClientScript is self-explanatory, and takes a Boolean value specifying whether or no client-side validation should be used if possible. If this value is True (which is the default), then if the client browser does support client-side script, then validation will be done on the client side), but if this value is False, then regardless of what the client browser is, the validation will always only be done on the server side.

Display Property

The Display property must be assigned to a value from the System.Web.UI.WebControls.ValidatorDisplay enumeration. The enumeration includes three values:

- None
- Static
- Dynamic

By default the value of the property is Static. This property defines how the error message is displayed when the validation fails. A value of None will result in the error message not being shown. Generally this will not be used unless a ValidationSummary control is in place, because the user must be informed of their error. Static and Dynamic are fairly similar, but there is one important difference between them. Firstly, the difference between Static and Dynamic does not have anything to do with how the client-side validation script works, or when validation events are triggered. When Static is chosen, the error message, even when not visible, takes up a physical portion of the screen where the message would be displayed when it becomes visible. Dynamic however, does not use up screen space when not visible, but when it is displayed, it shifts all the elements on the page down so that it can be displayed.

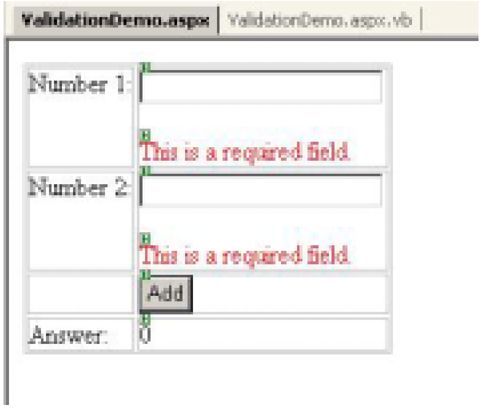
> From a more technical perspective, when Static is chosen, the 'visibility' CSS property is used to hide the error message, whereas when Dynamic is chosen, the 'display' CSS property is used.

Page.IsValid

If the form in a page that includes validators and has "invalid" controls, is submitted, either because scripting was not enabled in the client browser, or the validation controls were not set to use client-side validation, then it is important that the server-side code is aware of whether or not the submitted form is "valid". Even if a control, or controls, on the form are not submitted with suitable values (in the case of the RequiredFieldValidator, any value for a validated control would be suitable, so long as it's not an empty string, however this is not always true for the other validation controls, which will be introduced shortly), the event handler for the button, or other control used to submit the form, will still be fired. Therefore a mechanism must be in place to ensure that the code knows whether the form submission included invalid control values or not, otherwise the entire objective of using validators would be undermined.

For example, if a form included two textboxes, a button and a label, and this form would add the two numbers in the textbox when the button was clicked and output the result to the label, the button's Click event handler needs to know that values have been entered in the textboxes. RequiredFieldValidators are used to ensure that values are entered, but if client-side validation is circumvented, the button event will still fire when the form is submitted, and an exception will be thrown because an empty string cannot be converted to an integer. To check that the submitted form does not include invalid controls, the Page object's IsValid property should be used. This property returns a boolean value of True if the submitted form does not include any invalid controls, and False if it does. An If condition should be wrapped around the code inside the button's Click event handler ensuring that the Page.IsValid property is True.

Figure 7.2 shows the UI of the above example, with Listing 7.1 showing the button's Click event handler.



[Figure 7.2 – Page.IsValid example form]

```
 Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSubmit.Click
  If Page.IsValid Then
   lblAnswer.Text =
System.Convert.ToString(System.Convert.ToInt16(txtNumber1.Text) +
System.Convert.ToInt16(txtNumber2.Text))
  End If
 End Sub
```

[Listing 7.1]

The TextBox controls are named txtNumber1 and txtNumber2 respectively. The Button is btnSubmit and the Label holding the answer is lblAnswer. When btnSubmit is clicked, the validating controls should ensure that txtNumber1 and txtNumber2 do have values inserted. If the form is submitted when txtNumber1 or txtNumber2 do not include a value, the Page.IsValid property will be False when the button's event handler is called. If it is True, the textboxes must contain values, in which case lblAnswer's Text property is set to the sum of the values contained in txtNumber1 and txtNumber2.

Regardless of whether the submitted form includes invalid controls or not, the Page.IsValid property will always be true when read in the Page_Load event handler, as it has not yet been correctly set by the server, and should therefore not be relied upon in event handlers for the Page's OnLoad event.

CompareValidator

Fields on a form often need to be compared to one another, or specific values to ensure that the user is entering acceptable data. Possibly the most commonly used example of where a CompareValidator could be implemented is in a registration form, where the user must enter their chosen password twice. Another common scenario where fields must be compared to each other is entering an email address, then entering it a second time as a confirmation. The CompareValidator also lets you ensure that one field's value is larger than, or less than another field's. This ability could potentially be used when a user is asked to enter in a range of dates, where the start date must obviously be before (i.e. less than) the end date.

The ability of the CompareValidator to compare the value of a control to a specific defined value also has numerous uses. It could be used to ensure that a value is larger than 0. Perhaps the user cannot set a date before the current date. Conversely, the user might not be allowed to enter a value after a specific date.

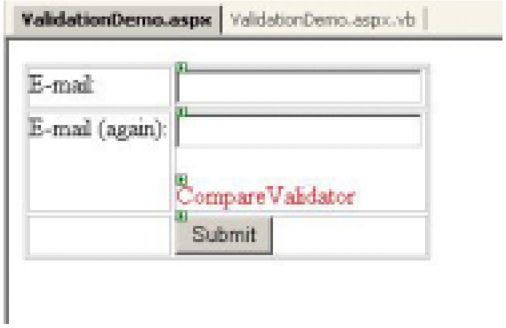
> If the value entered by the user must be bigger than X, but smaller than Y, then rather than using two CompareValidator controls, the RangeValidator should be used.

However, possibly the most useful feature of the CompareValidator is its ability to ensure that the value entered by a user is of the correct data type. For example, if an Integer is expected, then using the CompareValidator to ensure that indeed an Integer value is entered mitigates the need to include error checking code when converting the control value to an Integer in the server-side code when the postback occurs.

Comparing Control Values with each other

The CompareValidator can be used to ensure that one control's value relates to another's in a specific way. It can ensure that the value is equal to, not equal to, greater than, greater than or equal to, less than or less than or equal to another control's value. This functionality may typically be used in registration forms, where a confirmatory value may be entered for specific fields, such as passwords, or email addresses. To demonstrate this, perform the following steps:

1. Create a new web form
2. Add labels, textboxes, a button and a CompareValidator as shown in figure 7.3.



3. Name the controls lblEmail, lblEmailConfirm, txtEmail, txtEmailConfirm, btnSubmit and valEmailConfirm respectively.
4. Set the ErrorMessage property of valEmailConfirm to "E-mail addresses must be the same."
5. Set the ControlToValidate property of valEmailConfirm to txtEmailConfirm.
6. Set the ControlToCompare property of valEmailConfirm to txtEmail.
7. Compile the project and load the Web Form. The validator control should not be visible.
8. Enter in two different email addresses and click "Submit". A message should appear informing you that the email addresses must be the same.

9. Enter in two identical email addresses, click “Submit”, and the warning should disappear.

10. Enter an email address into the first textbox, and leave the second textbox blank, then click “Submit”. No error should be displayed.

Implementing the CompareValidator is very similar to using the RequiredFieldValidator, with only a few minor differences. The control is placed on the page, the error message is set and the control that it must validate is set. Since it is comparing one control’s value to another’s, the ControlToCompare property must also be set, and this property specifies which control the ControlToValidate control must be compared with. In this instance, all the default property values will suffice.

In instruction 10, a special behaviour (Microsoft did purposefully include this feature) of the CompareValidator is shown – when the value of the control to validate is empty, the control will pass validation. To prevent this, should you so wish, a RequiredFieldValidator can be added, and its ControlToValidate property also set to txtEmailConfirm. This will ensure that an empty string is not allowed, and with validator will cause the form to be invalid.

Again, as with the RequiredFieldValidator, and all the other validation controls, client-side scripting is used if possible (assuming that the EnableClientScript property is set to True), and if the form is submitted despite it not being valid, the Page.IsValid property will be set to False. The use of the Display property is also identical to its use with the RequiredFieldValidator, and all the other validation controls.

Comparison Operators

The default behaviour of the CompareValidator control is to ensure that the values compared are equal. However, the Operator property of the control allows a variety of operators to be chosen. The property must be assigned a value from the System.Web.UI.WebControls.ValidationCompareOperator enumeration. The enumeration includes the members shown in Table 7.1.

Member	**Description**
DataTypeCheck	Ensures that the value is of a specific type (dealt with in the section “Comparing Control Values with Data Types”)
Equal	Ensures that the relevant values are equal
NotEqual	Ensures that the relevant values are not equal
GreaterThan	Ensures that one value is greater than the other
GreaterThanEqual	Ensures that one value is greater than or equal to the other
LessThan	Ensures that one value is less than the other
LessThanEqual	Ensures that one value is less than or equal to the other.

[Table 7.1 – ValidationCompareOperator Enumeration Members]

In previous example, setting the Operator property of the valEmailConfirm control to NotEqual will result in the error message being displayed if two equal strings are entered into the textboxes. Since VB.NET can compare strings using the inequality operators, setting the Operator property to GreaterThan will result in the error being displayed is the first textbox’s value is B and the second’s is A, for example. However, the Operator property’s values other than ‘Equal’ only really become useful when used with data types other than strings.

Value Data Type

HTML form controls are typeless by nature, and they therefore assume the type “string” in .NET, since that is the data type that they most often contain. For the most part, this is perfectly acceptable and does not pose too much of a problem. However, with validation, the data type is often important. For example, if a CompareValidator were used to ensure that an integer value entered into one textbox was larger than one in a second textbox, the desirable result would be obtained if the value of the first textbox was 9 and the value of the second was 8. However, if the value of the first was 10, and the value of the second was still 8, the CompareValidator would incorrectly claim that the form was invalid, because when comparing strings, the first character is compared against the first character, and the comparison only continues to the second character if the first characters were the same. Thus, in a string comparison, 8 is larger than 10. This is obviously not the desired result, and the validator needs to be informed that it must perform its comparisons as Integer comparisons, not as string comparisons.

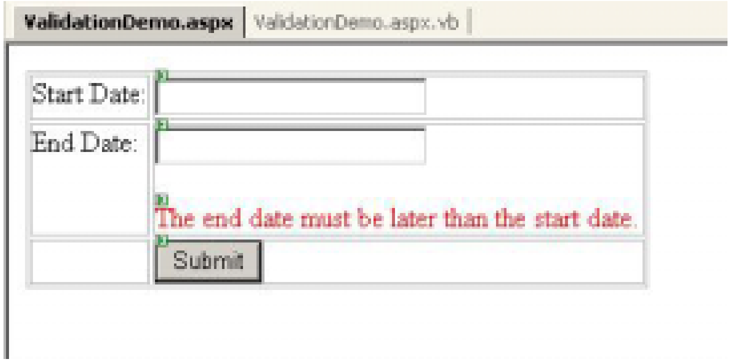
To this end, the CompareValidator includes the Type property, which accepts a value from the System.Web.UI.WebControls.ValidationDataType enumeration. Table 7.2 lists the enumeration members.

Member	**Description**
Currency	Casts the values to the Currency data type before comparing.
Date	Casts the values to the DateTime data type before comparing.
Double	Casts the values to the Double data type before comparing.
Integer	Casts the values to the Integer data type before comparing.
String	Casts the values to the String data type before comparing.

[Table 7.2 – ValidationDataType Enumeration Members]

Figure 7.4 shows a form that allows the user to enter a start and an end date. The CompareValidator has the following properties set:

- ControlToCompare: txtStartDate
- ControlToValidate: txtEndDate
- Operator: GreaterThan
- Type: Date



The end date must be later than the start date.

[Figure 7.4]

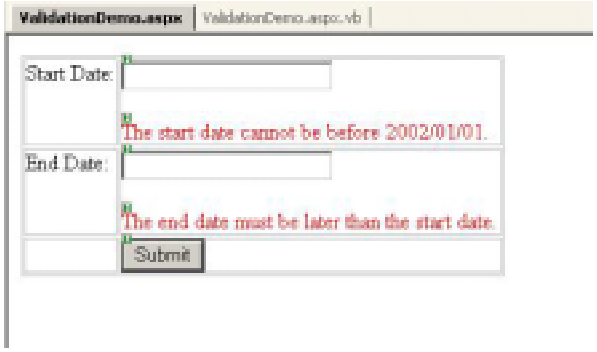
In this case, the CompareValidator will compare the values entered as dates, and ensure that the end date is a later date than the start date.

Comparing Control Values with Static Values

While the CompareValidator can be used to ensure that control values have a specific relationship with other control values, it can also be used to compare a control’s value with one static value that is specified. Extending the start and end date example started in the previous section, a CompareValidator could be set to validate the start date field to ensure that the date entered is not before the current date. The CompareValidator can be used in all forms where fields have a limit of how low, or high, a value can be (but not both – the RangeValidator should be used in such situations). For example, on a product ordering form, the quantity field may not be less than 1, but it does not have a maximum.

When using the CompareValidator to validate a control’s value against a static value, the ControlToValidate and ValueToCompare properties must be set accordingly, but the ControlToCompare property must not be set. The ValueToCompare property is a string property, so values assigned to it must be strings. If another data type is chosen using the Type property, then during validation both the control value and ValueToCompare will be cast as the chosen data type before being compared.

Figure 7.5 shows a continuation of the example in the previous section, shown in Figure 7.4. A new CompareValidator has been added to ensure that the start date is a date equal to or after 2002/01/01.



The start date cannot be before 2002/01/01.

The end date must be later than the start date.

[Figure 7.5]

The first CompareValidator had the following properties set:

- ControlToValidate: txtStartDate
- Operator: GreaterThanEqual
- Type: Date
- ValueToCompare: 2002/01/01

The form will be invalid if a date earlier than 2002/01/01 is entered, but any date including and after 2002/01/01 will be valid.

Programmatically Setting the ValueToCompare Property

Programmatically assigning a value to the ValueToCompare property can be particularly useful. Whereas when the CompareValidator is used to compare the values of two controls with each other the values are dynamic, the comparison of a static value is not intrinsically dynamic. However, since ASP.NET allows the properties of controls to be modified at run-time, the ValueToCompare property can become dynamic. With the full power of the ASP.NET framework at hand, the ValueToCompare value can be set to a value from a database, for example. Again, following on from the start and end date example, if that form was part of the booking process for an event, a database might contain a value of when the last event was booked for, and therefore the start date in the form can be validated to ensure that it is at least 5 days after the last event finished. As another example, the start date might be validated to ensure that it is after the current day's date. This obviously makes the ValueToCompare "method" significantly more effective and useful in some scenerios than if only 1 value is ever used.

The ValueToCompare property should be assigned in the Page's Load event handler. Assuming that a validator valDateStart exists, and has its other properties set appropriately, the following code will ensure that the date in the ControlToValidate property is not before the current date:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
 valDateStart.ValueToCompare = Now.Date.ToString("d")
End Sub
```

> When dealing with the Date data type in the CompareValidator, it is very important to ensure that the correct date format is used. The correct format's positioning of the year, month and day elements is largely dependant on the server's regional settings, but whatever the YMD format, the short date should be used, and a time should not be included. This is the reason for the previous example specifying the string parameter "d" in the Date's ToString() method – it ensures that the function returns the date in short format, without a time.

This ability lends even more power and flexibility to the CompareValidator. The ValueToCompare property can now be assigned a value from any server resource, from an event log entry, to a node in an XML document, to a value from a SQL Server database.

Validating Data Types

Possibly one of the greatest causes for the requirement of error checking on the server-side has traditionally been the fact that users will often enter data in an incorrect format. For example, when an Integer is required, they might enter a floating-point number, which, if unchecked, will result in an exception being raised when the conversion from the string from the control to an Integer occurs. Likewise spaces or commas might be used to separate groups of numbers, which would again cause errors when the type conversions take place in the server-side code.

The CompareValidator's Operator property can be assigned one member of the ValidationCompareOperator enumeration that is the odd one out, so to speak – the DataTypeCheck operator. When this operator is chosen, only the ControlToValidate property should be set – neither the ControlToCompare or ValueToCompare properties need to be used. As the name suggests, the DataTypeCheck operator ensures that the data in the ControlToValidate control's value is entered in a suitable format, such that it can be successfully converted to a specific data type. The data type that this "mode" of the CompareValidator checks against is specified in the Type property, which accepts a value from the ValidationDataType enumeration – "String", "Integer", "Date", "Currency" or "Double".

For example, to ensure that the value entered into a textbox, txtValue, is a date, a CompareValidator should be inserted onto the form, and have the following properties set:

- ControlToValidate: txtValue
- Operator: DataTypeCheck
- Type: Date

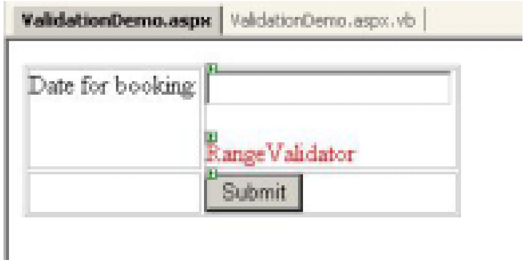
This would ensure that a value that can be cast from string to DateTime is inserted into the txtValue control.

RangeValidator

The RangeValidator is, in many respects, quite similar to the CompareValidator when it uses the ValueToCompare property. The RangeValidator allows a specific range of values in a control to be enforced. A simple example would be a field that requires a value from 1 to 10 to be entered. A RangeValidator control could be used to ensure that only Integers values from 1 to 10 are allowed. As with the CompareValidator, the RangeValidator can compare and validate values using the String, Integer, Double, Date and Currency data types. The RangeValidator could therefore also prove useful for entering in amounts of money (nothing less than 1 unit, but no more than 1000), or for a booking sheet, where dates before the current cannot be chosen, but the booking cannot be more than 4 weeks in advance.

To demonstrate the capabilities of the RangeValidator, perform the following steps:

1. Create a new web form.
2. Add a label, textbox, button and RangeValidator as shown in figure 7.6.

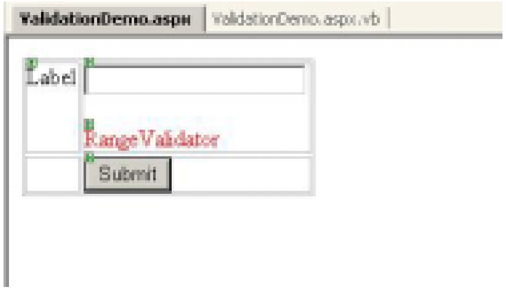


3. Name the controls lblBooking, txtBooking, valBooking and btnSubmit respectively.
4. Set the ErrorMessage property of valBooking to “Please enter a date between 2002/01/01 and 2002/02/01.”
5. Set the ControlToValidate property of valBooking to “txtBooking”.
6. Set the Type property of valBooking to “Date”.
7. Set the MinimumValue property of valBooking to “2002/01/01” and the MaximumValue property to “2002/02/01”.
8. Compile the project and load the Web Form.
9. Enter in a date – any date between the 1st of January 2002 and the 1st of February will be allowed, but any date outside that range will result in the validator displaying an error message.

As with the CompareValidator, and the remaining validator controls, if no value is entered the control will pass validation – to prevent this, a RequiredFieldValidator must again be used. The RangeValidator’s Type property allows multiple different data types to be compared, and as such the RangeValidator should provide suitable functionality for most situations where a value must be bound within a specific range.

The MinimumValue and MaximumValue properties can be set at runtime in the Page_Load event handler, making the RangeValidator more flexible. These values could be read from a database, a text file, XML, or even a Web Service.

Figure 7.7 shows a form with a label, textbox, RangeValidator and button added, named lblNumber, txtNumber, valNumber and btnSubmit respectively. The validator is set to validate txtNumber.



[Figure 7.7]

Listing 7.2 shows the Page_Load event handler for the form, which requires the user to enter a number in a randomly chosen range, which is displayed in the label.

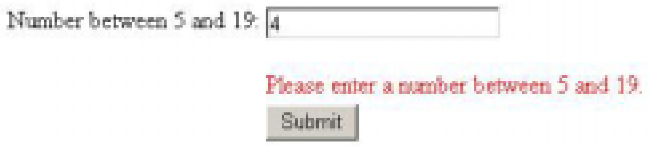
```
 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  Dim intMin, intMax As Integer
```

```
    Dim objRandom As New System.Random()
    intMin = objRandom.Next(1, 10)
    intMax = objRandom.Next(11, 20)
    valNumber.MinimumValue = intMin.ToString()
    valNumber.MaximumValue = intMax.ToString()
    valNumber.ErrorMessage = "Please enter a number between " &
   intMin.ToString() & " and " & intMax.ToString() & "."
    lblNumber.Text = "Number between " & intMin.ToString() & " and " &
   intMax.ToString() & ":"
   End Sub
```

[Listing 7.2]

The System.Random class provides the Next() method to generate random numbers. The Next() method has several overloaded versions, one of which allows two Integer parameters to be passed specifying the range that the random number must fall between. intMin is therefore assigned a value between 1 and 10, and intMax is assigned a value between 11 and 20. The validator's MinimumValue and MaximumValue properties are then set using these randomly generated numbers. Since these properties are string properties, the Integers must first be converted using the ToString() method. The validator's ErrorMessage is then set appropriately, and finally the label is set to inform the user of the range that they must enter a value between.

Figure 7.8 shows what the output may look like:


Number between 5 and 19: 4

Please enter a number between 5 and 19.

[Figure 7.8]

The ability to programmatically modify the RangeValidator's properties lend an even greater flexibility to the control, especially when combined with database access, and other functionality offered by the .NET Framework.

RegularExpressionValidator

Of all the validating controls, the RegularExpressionValidator is probably the most powerful and most flexible, with the exception of the CustomValidator. However, the power of the RegularExpression validator is in essence not owed to itself, but rather to regular expressions, as the name suggests. Basically the RegularExpressionValidator forces a control's value to conform to a regular expression. This allows the validator to ensure that almost any string is entered in the correct format, from telephone numbers, to email addresses to SSN's. To continue, a rudimentary knowledge of what regular expressions are, and how they work is required.

Regular Expressions Primer

Regular Expressions is an entire topic in itself, and entire books have been written solely on the topic, so this primer is far from being a complete guide to the subject matter, but should prove sufficient to leave you at least knowing the basics of string matching using regular expressions.

Regular Expressions can be thought of as a highly specialised programming language for dealing with strings. The two functions that regular expressions can perform are ensuring that a string matches a particular format, or extracting a substring from a string that matches a particular format. In a regular expression, you specify the format that you want a string to match. This is basically done by providing a set of rules that a string must comply with in order for it to “match”. For the purposes of this primer, only string matching will be demonstrated.

> The System.Text.RegularExpressions.Regex class provides full regular expression functionality for matching and extracting string matches on any string. See the .NET SDK documentation for more information.

The regular expression’s rules are laid out in a string called a pattern. The most basic pattern is one that matches a string, character for character. For example, the pattern “test” will match the string “test”, and no other string. However, there is a list of so-called “metacharacters” that don’t match themselves. These characters provide special functionality to the regular expression and generally affect the way that other parts of the regular expression function. The metacharacters are:

. ^ $ * + ? { [ ] \ | ( )

Firstly, the square braces (‘[‘ and ‘]’) - these are used to denote a character set, or a list or range of characters that can be matched. For example, the pattern “[abc]” will match with any of the strings “a”, “b” or “c”. The – character can be used to specify a range. Thus the pattern “[a-c]” is equivalent to the pattern “[abc]”. The regular expression that matches with any lowercase alphabetic character is “[a-z]”.

The metacharacters do not apply their special functionality when used within classes. For example, the pattern “[xyz?]” will match with the strings “x”, “y”, “z” and “?”. The question mark is a metacharacter, but since it is included in a set, it’s special meaning does not apply.

The caret (‘^’) can be used with in a set to specify that the pattern must match any character(s) except those specified in the set. The caret must be the first character in the set – if it is included elsewhere it will simply be matched like any other character. The pattern “[^abc]” will match with any one character string that is not “a”, “b” or “c”.

The backslash (‘\’) acts as an escape character, and allows metacharacters that would normally perform a specific function to be matched as if they were normal characters. The backslash must simply proceed the metacharacter that must be escaped. The pattern “\[“ will therefore match the string “[“, as will “\\” match “\”.

The backslash is also used to specify pre-defined sets. These are commonly used sets that can be specified as shown in table 7.1.

Pattern syntax	Description	Equivalent Set

\d	Matches any decimal digit	[0-9]
\D	Matches any non-digit character	[^0-9]
\s	Matches any whitespace character	[\t\n\r\f\v]
\S	Matches any non-whitespace character	[^ \t\n\r\f\v]
\w	Matches any alphanumeric character	[a-zA-Z0-9_]
\W	Matches any non-alphanumeric character	[^a-zA-Z0-9_]

[Table 7.1 – Regular expression character set shortcuts]

These escapes can also be used within sets. “[\w]” accepts any alphanumeric character, or a space. “[\w\s]” accepts any alphanumeric character, or any whitespace character. Likewise, these escapes can also be used outside of sets. “\w\s” matches with any string where its first character is an alphanumeric character, and its second is a whitespace character.

The period, or fullstop (‘.’) metacharacter is used to specify any character, except a newline. Therefore the pattern “a.b” will match with any 3-character string starting with “a” and ending with “b”, so long as the 2nd character isn’t a newline character. Thus “a1b” will match, as will “a b”, as will “a|b”. However, “a c” won’t match (the 3rd character is “c”, not “b”), and neither will “c b”.

Some of the most important rules that can be set in regular expression patterns are “repeating rules”. Regular expressions can be created that allow a string to repeat a character, or set of characters, a specific number of times. This is where regular expressions really set themselves apart from simply string manipulation functions.

The asterisk (‘*’) metacharacter is used to specify that the character, or character from a character set, before it can be used any number of times. Thus “ab*c” will match with “ac” (no “b”’s), “abc” (1 “b”), “abbc” (2 “b”’s) etc. This functionality also applies to character sets, so the pattern “a[xy]*c” will match with “ac”, “axc”, “axyc”, “axxc”, “axxyc” etc, but not “axy”, because it doesn’t end with “c”, and not “abc”, because “b” is not in the specified set. The pattern “.*” should match with any string, as it allows any character to be specified any number of times. This can also be compounded with regular characters, and for example the pattern “.*@microsoft.com” should allow any string ending with “@microsoft.com” to be specified.

The plus (‘+’) metacharacter is very similar to the asterisk, but has one subtle difference. It allows a character, or a character from a character set to be specified one or more times. The asterisk allows the character not to be entered at all, but the plus metacharacter requires at least one instance of the character to appear. Therefore the pattern “ab+c” will match with “abc” and “abbc”, but not “ac”. Again, the plus metacharacter can also apply to sets, and can also be used in conjunction with other characters or metacharacters in a complex pattern.

The question mark (‘?’) metacharacter allows a character to be specified either once, or not at all. It effectively marks a character or set as optional. Therefore the pattern “Micro-?soft” will match with both “Microsoft” and “Micro-soft”.

The curly braces (‘{‘ and ‘}’) allow a specific range of repetitions of a specific character or character set to take place. The minimum and maximum number of repetitions is specified within the braces, separated by a comma. The pattern “.{3,6}” matches with any string (provided that it doesn’t include any newline characters) between 3 and 6 characters long. The string must not be less than 3, or more than 6 characters long. This notation can also be used to specify only a minimum or maximum number of repetitions by omitting one of the values. Omitting the minimum will result in a minimum of 0 being assumed, and omitting the maximum will result in a maximum of infinity being assumed. Therefore the pattern “.{3,}” will match with any string 3 characters or longer, whereas the pattern “.{,3}” will match with any string less than or equal to 3 characters in length. The last function that the curly braces can perform is to specify a specific number of repetitions for a character or character set. This is done by only supplying one value, with no comma. The pattern “\d{5}” will therefore match with any 5-digit decimal number. The string must only contain digits, and must be exactly 5 characters in length.

The brackets (‘(‘ and ‘)’) are used for grouping characters and sets together. When used together with other metacharacters, this can be used to create rules that would otherwise not be possible, or more difficult to implement. For example, the pattern “(do*g)?s” specifies that the “do*g” rule can be omitted, since it is grouped using brackets and modified using a question mark. However, if the user decides to include the string inside the brackets, it must be correctly formatted. In either case, the trailing “s” is obligatory. Therefore “dogs” will match the regular expression, as will “dgs” or “doogs”. “s” will also match (it is omitting the rule in the brackets), but “ds” will not (it is missing the required “g”), and neither will “oogs”. “dog” will also fail to match. However, if the regular expression omitted the brackets and was simply “do*g?s”, the “d” would be required, as would the “s”, and only the “g” would be optional.

The vertical bar (‘|’) is used to specify an “or” condition - either one part of the expression can be completed, or another part can be completed. The vertical bar is normally used with brackets to ensure that the “or” is applying to the correct parts of a pattern and for the purpose of clarity for the programmer. For example, the pattern “Hello, \w*|Hi, \w*” specifies that the string must either begin with “Hello, “ or “Hi, “, and can then contain any number of alphanumeric characters, but to clarify its meaning, it would be better written as “(Hello, \w*)|(Hi, \w*)”. For that pattern “Hello, Lionel” would match, as would “Hi, Lionel”, but “Hello Lionel” would not, and neither would “Hi Lionel” (in both cases, the comma is missing).

Building and Deciphering Regular Expressions

Armed with this knowledge of regular expression fundamentals, you should now be able to both dissect and dismantle almost any regular expression, and build your own regular expressions to match almost any string format imaginable. At first glance, regular expressions can seem incredibly complex and cryptic, but by breaking them up into smaller parts and individually identifying what each part of a regular expression does by applying the rules, it becomes fairly trivial to interpret the functionality of any given pattern. Likewise when building regular expressions, the task should be broken down into small steps, which makes the whole process significantly easier.

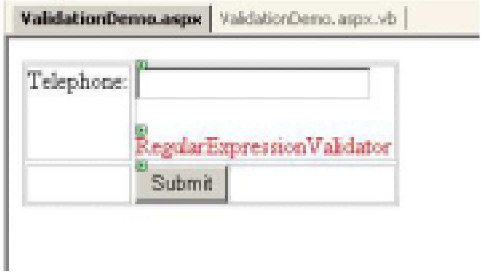
Regular Expressions and the RegularExpressionValidator

If the relative complexity of regular expressions is ignored, the RegularExpressionValidator is actually a fairly simple control. It allows a

ControlToValidate property to be set, specifying the control that it must validate, and the ValidationExpression property, that specifies the regular expression pattern that the ControlToValidate's value must match. If the value matches, then the control will be successfully validated, but if the value does not match the pattern, the control will be marked as invalid, thereby invalidating the form. As with the other validation controls, the RegularExpressionValidator will always successfully validate a control with an empty value, and to prevent this a RequiredFieldValidator must be used.

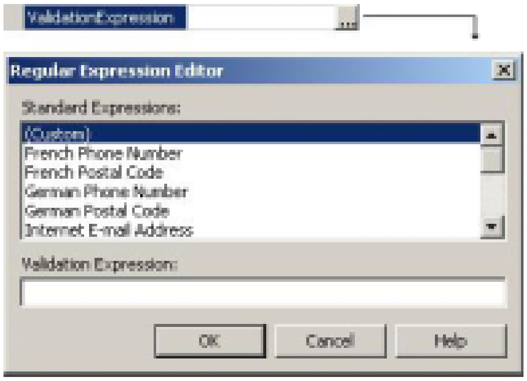
To demonstrate the use of the RegularExpressionValidator to ensure that a telephone number is entered in the correct format, perform the following steps:

1. Create a new web form.
2. Add a label, textbox, RegularExpressionValidator and button, as show in figure 7.9.



[Figure 7.9]

3. Name the controls lblTelephone, txtTelephone, valTelephone and btnSubmit respectively.
4. Set the ErrorMessage property of valTelephone to "Please enter a telephone number in the format (000) 123-4567, 000-123-4567 or 123-4567."
5. Set the ControlToValidate property of valTelephone to "txtTelephone".
6. Click on the button in the properties window when the ValidationExpression property is selected. The "Regular Expression Editor" dialogue should appear, as shown in Figure 7.10.



[Figure 7.10 – Regular Expression Editor]

7. Scroll down to the "U.S. Phone Number" option and select it. A regular expression should appear in the textbox below. Click "Ok".
8. Compile the project and load the web form.

9. Try entered phone numbers in numerous formats. Only the three formats mentioned in the validator’s error message will be allowed.

The regular expression that the Regular Expression Editor was used to insert looks like this:

```
((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}
```

This is one of the many pre-defined regular expressions that are available for common formatting tasks, such as e-mail addresses, postal codes, and in this case, phone numbers. However, the defaults may not be suitable, so it is possible to define a custom regular expression for your needs. This can either be based on one of the pre-defined patterns, or started afresh. To continue our example of requiring that a US phone number is entered we could extend the regular expression so that it requires that US’s International code also be entered (+1). “\+1 ” can be added to the expression to achieve this, so the final pattern is:

```
\+1 ((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}
```

To make this change, select the ValidationExpression property in the properties window, click the button that appears and add “\+1 ” to the regular expression that is currently in the text box. You will notice that the selection in the listbox changes to “(custom)”. Click “Ok”, and the change is made. The validator will now only accept numbers in the format +1 (000) 123-4567, +1 000-123-4567 or +1 123-4567.

Conclusion

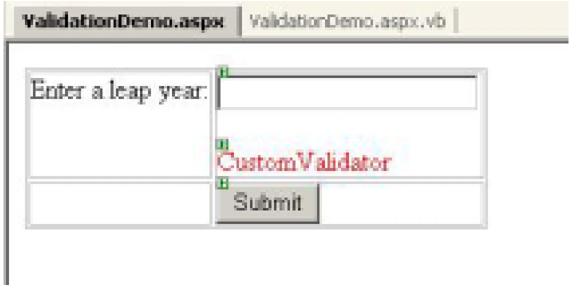
The RegularExpressionValidator is a very powerful tool that can be used to ensure that a string conforms with almost any format imaginable, from standard phone number formats, to email addresses to ISBNs. The use of the RegularExpressionValidator can be very useful in ensuring that data captured from users is all in the same format, which makes analysing and manipulating it later on significantly easier.

CustomValidator

Although the functionality provided by the previous validator classes is very comprehensive, situations do exist where a validation is required that cannot be achieved by any of the previous controls. For such situations, the CustomValidator control is provided, which allows you to build your own client- and server-side validation methods for a specific control. It essentially lays the foundations for a validator, and lets you specify exactly what criteria the control to be validated must conform to.

An example of a scenario where a regular validator would not provide sufficient functionality to correctly validate a control is if a leap year must be entered, and the validator must ensure that a year that is a leap year is entered. To build a form that does this, perform the following steps:

1. Create a new web form
2. Add a label, textbox, CustomValidator and button, as shown in figure 7.11.



[Figure 7.11]

3. Name the controls lblLeapYear, txtLeapYear, valLeapYear and btnSubmit respectively.
4. Set the ErrorMessage property of valLeapYear to “Please enter a leap year.”
5. Set the ControlToValidate property of valLeapYear to txtLeapYear.
6. Add the event handler in Listing 7.3 to the web form’s code behind class:

```
Private Sub valLeapYear_ServerValidate(ByVal source As Object, ByVal args
As System.Web.UI.WebControls.ServerValidateEventArgs) Handles
valLeapYear.ServerValidate
  Try
    Dim i As Integer = Convert.ToInt16(args.Value)
      If (i Mod 4) = 0 Then
        'It is a leap year
        args.IsValid = True
      Else
        args.IsValid = False
      End If
  Catch e As Exception
    args.IsValid = False
  End Try
End Sub
```

[Listing 7.3]

7. Compile the project and load the web form.
8. Enter in a year that is not a leap year (such as 2002), and click “Submit”. The form will submit, but the postback will result in the validation error message being displayed.

As with all the other validation controls, if an empty string is entered, the control passed validation. This validator also affects the whole form’s validity, so if the control is not successfully validated, the Page.IsValid property is set to false.

The CustomValidator’s ServerValidate event is raised when validation is performed on the server. The event handler for the event receives two parameters, an object, and an instance of the System.Web.UI.WebControls.ServerValidateEventArgs class. This object’s Value property passes the value of the control to validate, and the

IsValid property, which specifies whether the control is valid or not, must be set inside the event handler.

To check whether the entered year is a leap year, the year is divided by 4, and if there is no remainder, the year is a leap year. The IsValid property is set accordingly, and the whole block is encased in a Try block to catch any exceptions that may occur when converting the passed value to an Integer.

There is however one noticeable omission from this control thus far, namely, client-side validation – the control is only validated after the form is submitted. However, the CustomValidator does provide for this functionality through the ClientValidationFunction property. This property specifies a client-side function that can be used to validate the ControlToValidate on the client side. The function that is specified must accept two parameters, similar to its server-side counterpart – the sender object, and an arguments object. Listing 7.4 shows the client-side script block that defines a function for validating a leap year.

```
<script language="javascript">
 function LeapYearValidate(sender, args)
 {
  if ((args.Value % 4) == 0)
  {
   args.IsValid = true;
  }
  else
  {
   args.IsValid = false;
  }
 }
</script>
```

[Listing 7.4]

Insert this code into the web form’s UI code, and set the valLeapYear control’s ClientValidationFunction property to “LeapYearValidate”. The Web Form will now validate the txtLeapYear control on the client-side if possible, but it will still always validate the control on the server-side, using the ServerValidate event handler code inserted earlier.

Conclusion

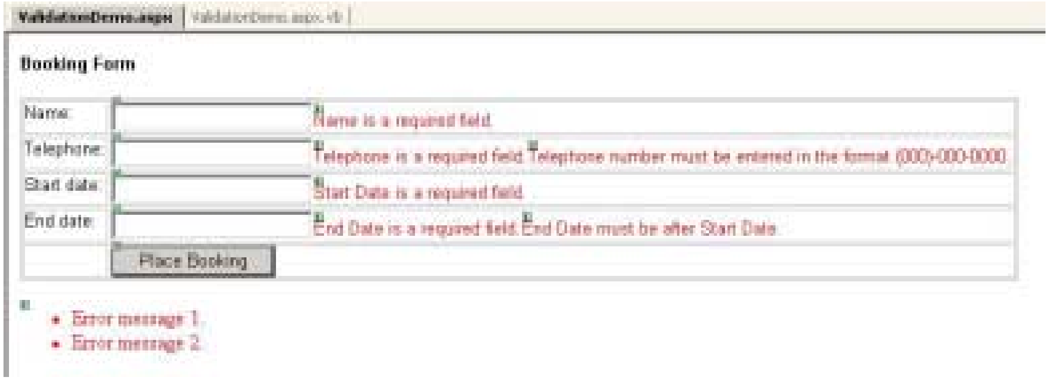
The CustomValidator is available to provide a method of validation almost any conceivable situation, and its ability it effectively only limited by the creativity of the programmer. It provides a solution when all the other validation controls are unable to offer suitable validation for controls in specific non-standard applications.

ValidationSummary

The ValidationSummary control isn’t actually a validator. Rather, it provides a mechanism that groups together all the incorrectly filled out controls on a form that didn’t successfully validate and lists all the errors that need to be corrected. This can be useful for placing at the top or bottom of a long form, so that users know exactly which controls need to be corrected, rather than having to scroll through the entire form checking to see which items haven’t been validated.

The use of the ValidationSummary is very simple indeed – in a form that includes validation controls, simply add a ValidationSummary control. It will automatically recognise the validators on the page, and display the validation summary accordingly at run time. The following steps demonstrate the use of the ValidationSummary control:

1. Create a new web form.
2. Add the controls and set the appropriate properties such that the form looks like Figure 7.12, and acts as expected.



Booking Form

Name: Name is a required field.

Telephone: Telephone is a required field. Telephone number must be entered in the format (000)-000-0000

Start date: Start Date is a required field.

End date: End Date is a required field. End Date must be after Start Date.

- Error message 1.
- Error message 2.

[Figure 7.12]

3. The final control on the form is a ValidationSummary control. At design time, its text will normally always be displayed as it is here. However, at runtime it will automatically recognise the validators on the form. The ValidationSummary control will operate without changing any properties, so all the properties can be left at their defaults.
4. Compile the project and load the web form.
5. Click “Place Booking”. All the RequiredFieldValidators should display their error messages, but the ValidationSummary should also provide a summary of all the errors that occurred at the bottom of the page.
6. Fill in the name field, and try to submit the form again. The ValidationSummary should change to reflect that the name field has now been correctly filled in.

The three properties unique to the ValidationSummary are DisplayMode, ShowSummary and ShowMessageBox. These three all deal with how the summary is displayed. The DisplayMode property takes a value from the ValidationSummaryDisplayMode enumeration. The members of this enumeration are shown in Table 7.2.

Member	Description
BulletList	Summary displayed in a bulleted list.
List	Summary displayed in a list.

SingleParagraph	Summary displayed in a single paragraph.

[Table 7.2 – ValidationSummaryDisplayMode enumeration members]

The BulletList member is the default value for the DisplayMode property.

The ShowSummary property is a boolean that specifies whether or not the summary should be displayed on the page at all. This would normally be used in conjunction with the ShowMessageBox property, otherwise it would negate the entire purpose of the ValidationSummary control by not displaying any summary.

The boolean ShowMessageBox property, which is by default false, specifies whether or not the validation errors should be displayed in a MessageBox (a JavaScript window.alert() dialogue). This can be useful for grabbing the attention of the user, and can be used on its own, by setting the ShowSummary property to False, or in conjunction with an on-page summary.

Validation Controls – The Definitive Solution to User Input Constraints?

The short answer to that question is no, validation controls are, in numerous situations, not the definitive solution to enforcing business rules and logical constraints on user input. However, validation controls are very useful in restricting the range of errors that a user can make before a form is actually submitted, and/or any important server-side code is executed. Validation controls benefit both the user and the programmer, because the user will normally be informed of any “simple” errors, such as accidentally skipping out a field, before the form is even submitted, thus saving them time. The programmer is also saved from the mundane task of checking to see if all the required fields have been filled out etc, and can worry more about enforcing more important business rules and logic. An example of a business rule might be that the date entered for a new booking on a booking form has to be after the current date. This type of more simplistic business rule can also be enforced using regular validation controls. However, there is a divide between what rules can and should logically be enforced using regular validation controls, and what rules make more sense to simply be programmed on the server-side in an event-handler or subroutine. Required fields, specific ranges and formats that must be adhered to and other such relatively simple rules can and should be enforced using validation controls, but continuing the example of the booking form where another business rule might be that only one person can be allowed to book for a day, so the user cannot enter a date that has already been booked for; this business rule should not be enforced using regular validation controls, but rather using server-side code, because the data is simply too complex to validate using only the requirements that the regular validators can enforce. In this case, the check to ensure that the date that the user has entered has not yet been booked for should take place in a server-side code block; either in the event-handler for the form’s submit button, or using only server-side validation in a CustomValidator control. Client-side validation is not feasible, because it would have to involve all relevant booking information for all the bookings being sent to the client in order for the validation to be accurate.

Another extremely important factor in applying input validation is that although client-side validation is useful from the perspective that it may often save the user a round-trip if they do fill in a field incorrectly, it should only be secondary to, and never be a substitute for server-side validation, which cannot be circumvented by the user. This is why all the regular validation controls have both server-side and client-side validation capabilities, and although they use client-side validation where possible, the server-side validation is always run to ensure that the user input is always validated. This is an important factor to take into consideration when using the CustomValidator, and if there is one cardinal rule for building custom validation using the CustomValidator, it is that you should start with the server-side validation, and then, if it makes sense, implement client-side validation.

For implementing more complex business rules that can only practically be enforced on the server-side, the CustomValidator is a very useful control, assuming that the business rule applies to one field only. However, if a rule applies to more than one field, the business logic should be embedded in the event handler or subroutine where the user input is being used.

In summary, the regular validation controls are useful for making the user experience more pleasant and making programming easier, but can only be used for applying simple business logic. The CustomValidator is inherently more advanced and can be used to apply more complex business logic, especially with regard to fields that rely on information already in data stores, but can only be effective where the business rule applies to one field only. For even more complex rules, plain server-side coding should be used to enforce the logic. In all the cases, client-side validation should be secondary as it is primarily for user convenience and its authenticity cannot be guaranteed, so server-side validation should always take place.

Conclusion

The ASP.NET validation controls provide a very wide range of validating functionality that is both easy-to-use, very effective and extremely flexible. By automatically combing client- and server-side validation capabilities, the controls ensure that the user obtains the quickest and best experience by using client-side validation when available, whilst still ensuring that fields are always entered correctly by using a mandatory server-side validation. The Page.IsValid property provides the necessary means to programmatically check whether a page has been submitted with valid controls or not, and the ValidationSummary control provides an easy solution for displaying a summary of the errors that the user needs to correct in a lengthy form.

Chapter 8

Data Programming with ADO.NET

9 times out of 10, if you're building anything larger than a small application, you will be using a database to store information. The Web is ultimately about the timely delivery of data to users, and one of the primary reasons that technologies such as CGI were developed was to make it easier for web developer to deliver data to the user, and to deliver it in a fashion that was simpler for the client to use and understand. In fact, one of Microsoft's first attempts at making the web easier to program was the IDC, or Internet Database Connector, technology whose sole purpose was to allow information from a database to be displayed on a web page, and for the user to be able to update it. Web programming technology has progressed significantly since then, but the real aim of the Internet remains to deliver interactive data to the user. ASP.NET provides a comprehensive set of class that encompasses all aspects of interacting with relational databases, such as Microsoft SQL Server. Dubbed as "ADO.NET", these classes are located in the System.Data namespace are the primary means for both Windows Forms and Web applications built using the .NET Framework to access data.

For those of you who don't come from a Microsoft programming background, ADO (ActiveX Data Objects) was Microsoft's data access technology that was used to access OLE DB and ODBC databases from any of their programming environments, including Visual Basic 6, Visual C++ 6, ASP and any other COM-compatible language or technology (such as Delphi). However, just as ASP.NET doesn't stand for 'Active Server Pages.NET' (it's just a name, and doesn't stand for anything), neither does ADO.NET stand for 'ActiveX Data Objects.NET'. ASP.NET is a technology that is completely new, yet builds on the strengths of ASP, and similarly, ADO.NET provides a completely new way of interfacing with databases whilst utilizing the best concepts from ADO. However, just as a dramatic mindset adjustment needs to be made when learning ASP.NET after having ASP experience, so too do you need to open your mind to a new way of thinking about data access when learning ADO.NET after having used ADO. In fact, if you haven't used ADO before, you're probably at an advantage because you won't have any misconceptions about how you expect things to work in ADO.NET.

I'd love to be able to tell you that ADO.NET is really easy to learn, and it won't take you longer than 10 minutes to fathom the changes Microsoft have made in their latest data access layer, but I'd be lying. In fact, for anyone who's had experience using ADO.NET's supposed predecessor (I say 'supposed', because in fact ADO.NET barely even resembles ADO in many ways), you're probably going to be thoroughly confused during your first read-through of this chapter, so I'm giving you fair notice now that you will more than likely need to read through and try out the examples in this chapter at least twice before you are comfortable with data access in .NET. Of course, once you've 'got it' and have broken out of the mould of traditional ADO thinking and see the rationale behind ADO.NET's way of performing certain tasks, you'll find that ADO.NET is indeed very easy to use and gives you a great deal of power and flexibility, which you'll wonder how you did without when you were using ADO. To make learning ADO.NET easier, you need to approach it with an open mind. Try to forget everything you know about ADO for the next hour while you’re reading this chapter, and while you’re reading, try to remember back to when you first learned ADO and muttered to yourself, “Why can’t this be simpler?” or “Why can’t I just do this?” because you’ll probably find that ADO.NET provides solutions to some of those questions, problems and general annoyances.

Changes in Microsoft Data Strategy

In the mid to late 1990s, Microsoft resolved to create a data access framework that would allow an application to interact with a wide variety of RDBMSs (Relational DataBase Management Systems) using exactly the same API. One the key parts in this framework was ODBC, or Open DataBase Connectivity, which was a technology that exposed a common view of a large number of database formats to any application consuming an ODBC data source. Besides the need to know the feature limitations of the database format (such as whether it supported stored procedures or not, for example), the application didn’t need to be concerned about which database was being used, since the API was identical for all databases and all that needed to change was the string defining which database to connect to. However, Microsoft seems to have diverged from their ‘Universal Data Access’ vision in .NET in quite a major way. First of all, there is no native ODBC support in .NET. Although it’s possible to download additional ADO.NET support for ODBC data sources or use a workaround (which will be discussed later), Microsoft are obviously trying to discourage the use of ODBC, and not without reason. However, ODBC isn’t the only existing Microsoft data technology to take a back seat in .NET. OLE DB, another database abstraction layer, plays a significantly lesser role in .NET as well. Whereas with ADO, OLE DB was the primary means (along with ODBC) of interfacing with compliant databases, in .NET Microsoft are trying to encourage a decidedly less ‘universal’ method of accessing data.

The rationale behind this shift in strategy is that even though having a single data access layer for a whole range of database systems is very convenient from a coding point of view (for example, it makes switching from one database system to another almost trivial in most cases, where all that needs to be changed is the connection string), this unified model does have a serious impact on performance, since the amount of layers don’t really allow for much specific optimization for each data source. In light of this, Microsoft tries to create a ‘best of both worlds’ environment in .NET. The next section will present the actual architecture of ADO.NET in a slightly more detailed fashion, but effectively what Microsoft have done is separate the data access layer from the data manipulation API. In ADO, you could either access an OLE DB or ODBC data source through one generic ADO connection object, and then execute queries and manipulate the data from there, all using the same set of objects regardless of the database system that is being used. However, in .NET the coupling between the API that is used to connect to and communicate with the database and the API that is used to work with the data after it has been retrieved has been loosened to the point that they are actually two totally separate sets of classes, and can only communicate between themselves because there is an internal standard that these classes must adhere to. Those ADO and database gurus amongst you may bring out the point that in fact the code that connects to the database and the code that manipulates data from the database are inextricably bound. However, ADO.NET negates this argument to a large extent because whereas ADO strongly encouraged “connected” access to data, where the application maintained a connection to the database for the entire duration that the data was being manipulated so the connection and manipulation had to be bound, ADO.NET strongly encourages the use of a “disconnected” view of data, where the data is all loaded into the application, the connection is then closed, and only then the data is manipulated. However, just as ADO encouraged connected data access, but still provided a means for a disconnected environment (implemented through disconnected recordsets), so too does ADO.NET provide a means to access data in a connected fashion if you’d prefer to do so.

In summary, Microsoft has changed their data access strategy from a ‘universal’ one-size-fits-all API model, to a more loosely coupled architecture. Secondly, ADO.NET strongly encourages the use of disconnected data in contrast to ADO, which encouraged the use of a connected implementation. With this in mind, we can move on to a discussion of how ADO.NET is structured.

ADO.NET Architecture

All the ADO.NET classes are found inside the System.Data namespace. The classes contained within it can, for our purposes, be broken up into two groups – the classes that are used for manipulating disconnected data from any data source, and the classes for communicating with external data sources.

Managed Data Providers

In ADO.NET, there exists a concept of ‘Managed Data Providers’, and essentially a Managed Data Provider is a set of classes that implement several interfaces in the System.Data namespace. These classes provide functionality to cover three main areas – connecting to the database, executing queries on the database (including stored procedures, if supported), and reading data from the database. .NET ships with two such providers, one for OLE DB data sources, and one specifically for SQL Server data sources. The advantage of separating the data access process was alluded to earlier, and the main reason is that of performance. When connecting to a SQL Server database through the OLE DB provider, you will receive significantly lower performance than if you connected using the specially built SQL Server provider, which bypasses the layers of OLE DB and connects directly to SQL Server.

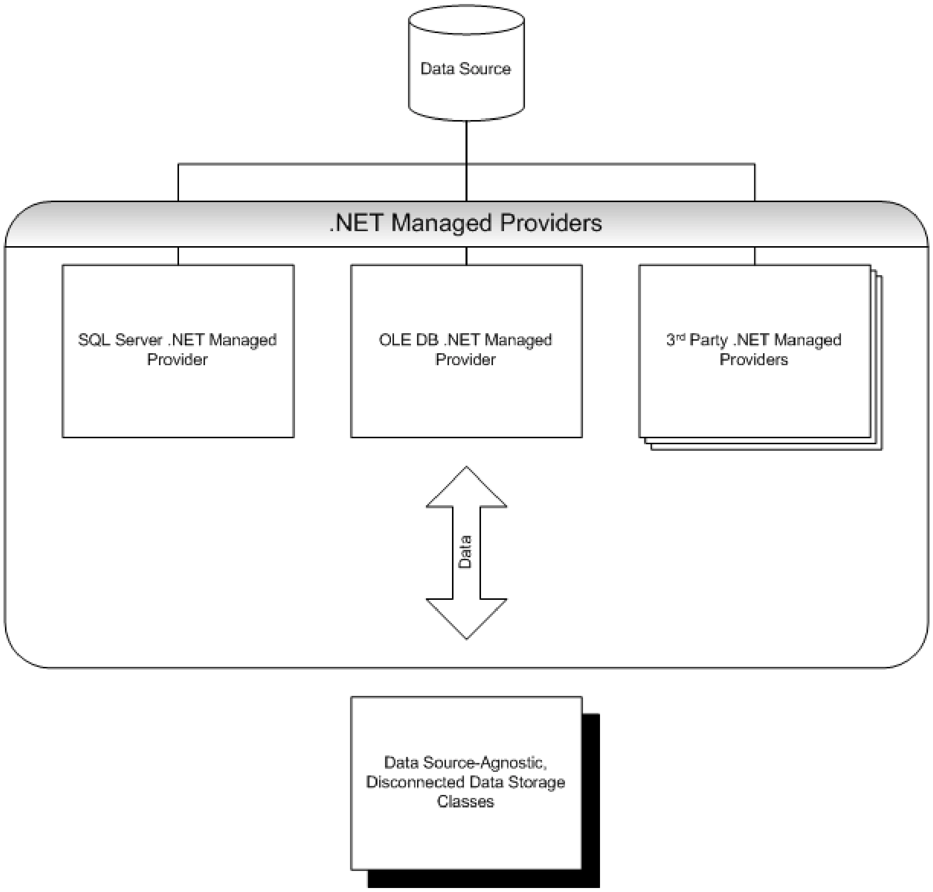
> Many figures have been quoted as to exactly how much better the SQL Manager Provider performs than the OLE DB Managed Provider. I conducted tests to settle the matter in my own mind, and although I certainly wouldn’t class my testing methodology as exhaustive and foolproof, I found that when a fair amount of data was being accessed, there was a 50% increase in performance of the SQL Server provider over the OLE DB provider, which is quite significant.

Although the only two Managed Providers that ship with .NET provide support for OLE DB and SQL Server data sources, Microsoft hopes that other major database vendors, such as Oracle, will develop Managed Providers for their own database products, thus allowing more efficient access than if the regular OLE DB provider is used.

> As was mentioned in the previous section, Microsoft doesn’t include direct support for ODBC in .NET – that is, an ODBC Managed Provider isn’t shipped with the .NET Framework. However, if you absolutely have to use ODBC, you can do one of two things. Firstly, you could download the additional ODBC Managed Provider from the MSDN website and use that, or alternatively you could use an OLE DB workaround. OLE DB allows you to access an ODBC source through it by using the “Microsoft OLE DB Provider for ODBC Drivers”, as apposed to the SQL Server or Jet OLE DB providers. Since using this method means that you won’t have to distribute the separate ODBC Managed Provider .NET Assembly, this workaround does have some advantages.

However, once data has been retrieved from a database using the appropriate Managed Provider, it will normally be transferred into the control of the database-agnostic “data manipulation classes”. I say, “normally,” because you don’t have to use the disconnected model, and can work in a connected fashion if it would be more appropriate in a given situation. Once this data has been transferred into the generic data-holding classes, the connection to the data source can be closed, and you are free to manipulate the data in whatever way you see fit. At the moment this doesn’t sound particularly useful or innovative, and you might even say, “But I can already do that with ADO,” but the real power lies not so much in the principal (which was indeed possible in ADO), but by the breadth of functionality implemented by the data-holding classes. The diagram in figure 8.1 shows the role that the Managed Providers play in ADO.NET.

Figure 8.1 – Managed Providers in ADO.NET


Data Source
.NET Managed Providers
SQL Server .NET Managed Provider
OLE DB .NET Managed Provider
3rd Party .NET Managed Providers
Data
Data Source-Agnostic, Disconnected Data Storage Classes

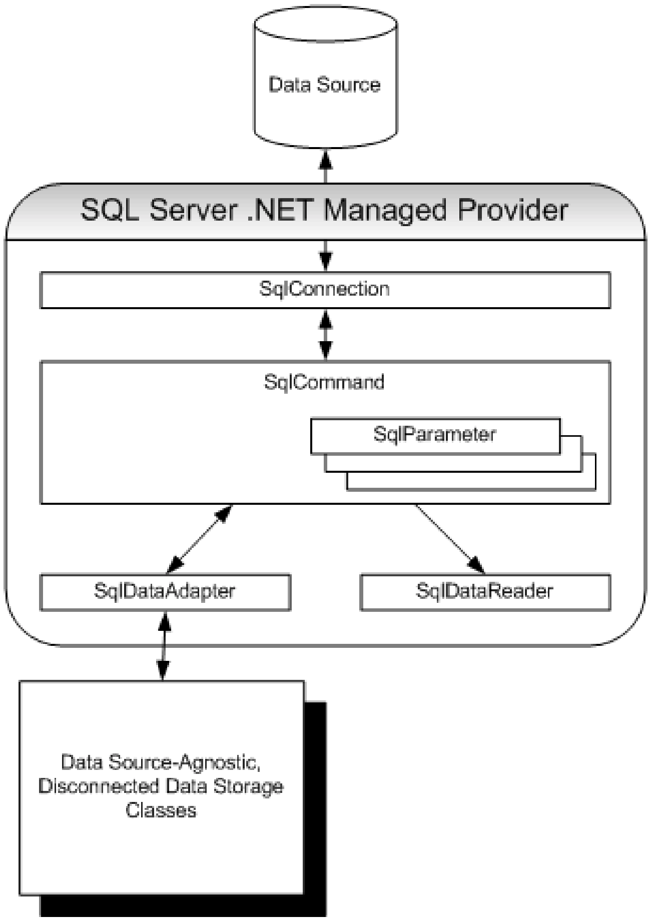
As you can see, the .NET Managed Providers communicate with the data source (be it a SQL Server database, an Access database or any other data store that can be accessed using OLE DB or one that additional Managed Providers can communicate with). The Managed Provider do that actual interfacing with the database, and (assuming you're going to use the disconnected model) send the data to the data storage classes. It's actually not a fluke that the bidirectional arrow is included inside the ".NET Managed Providers" block. This is because each Managed Provider contains a class that will fill the data storage classes and do updates on the data store when the data in the storage classes changes, which obviously involves reading data from, and writing data to the data source.

Up until now, we've only looked at the concepts of ADO.NET, and not at any of the classes that you'll actually be working with. Now that you've got a decent understanding of what a Managed Provider is, I'm going to show you the parts that constitute one by using the SQL Server Managed Provider as an example. The OLE DB provider has almost identical classes (just with different names), so you needn't worry if you don't use SQL Server. First up, all the classes that constitute the SQL Server provider are located in the System.Data.SqlClient namespace (the OLE DB ones are in System.Data.OleDb, and all start with OleDb instead of Sql). These are the most important ones:

* SqlCommand
* SqlConnection
* SqlDataAdapter
* SqlDataReader
* SqlError
* SqlException
* SqlParameter
* SqlTransaction

If you've used ADO before, some of these classes should sound vaguely familiar, such as SqlCommand, SqlConnection and SqlParameter. These three obviously map fairly closely to their ADO equivalents. However, there are also several new-looking classes, whose purpose isn't immediately clear from their names, such as the SqlDataAdapter and SqlDataReader. These two don't have direct ADO equivalents, and serve to fill newly created roles in the ADO.NET architecture. Figure 8.2 shows how the main classes in the SQL Server Managed Provider connect to communicate with the database.

Figure 8.2 – SQL Server Managed Provider Architecture


Data Source
SQL Server .NET Managed Provider
SqlConnection
SqlCommand
SqlParameter
SqlDataAdapter
SqlDataReader
Data Source-Agnostic, Disconnected Data Storage Classes

The SqlConnection Class

The SqlConnection class is the one responsible for communicating with the database. It creates the connection (and closes it), and allows the SqlCommand class to execute commands on the SQL Server. A SqlCommand object cannot operate without having an instance of a SqlConnection. The SqlConnection class works very similarly to its ADO counterpart, and allows you to specify a connection string (which is a string that normally specifies the IP/name of the SQL Server to connect to, the database to be used, and the authorization credentials, amongst other things). The connection string is normally specified as an argument in the class' constructor, and the connection can then be opened using the Open method, or closed using the Close method.

The SqlCommand Class

The SqlCommand is the object that is used to execute SQL queries on the SQL Server. It can execute text queries or stored procedures, hence the reason why SqlParameter objects are contained within it. A SqlParameter object represents (surprise, surprise), a parameter in either a text query or a stored procedure, and can represent either an input, output or return value. The SQL query text can either be assigned in the constructor, or using the CommandText property.

The SqlCommand object provides numerous methods for executing queries, depending on what kind of result you're expecting, and what you'd like to do with it. The four main methods are:

- ExecuteScalar
- ExecuteXmlReader
- ExecuteReader
- ExecuteNonQuery

ExecuteScalar is used when the result of an aggregate function is returned. An example of such a query might look like this:

```
SELECT Min(fieldname) FROM tablename
```

The ExecuteScalar method returns an Object (because effectively what it is doing is returning the first cell from the first row of results, and can be called for any query), and in the above scenario an Integer value might be returned.

ExecuteXmlReader returns an instance of the System.Xml.XmlReader class, which provides read-only, forward-only access to XML representing the result of the query. The ExecuteXmlReader method executes the query and converts the result to XML, which can then be read using the returned XmlReader class. This isn't a very commonly used method, and as such isn't reflected in figure 8.2 nor will it be detailed any further, but if you need to work with XML from a data source, you can keep in mind that this is one of your options.

ExecuteReader returns an instance of the SqlDataReader class, which can most closely be compared to an ADO read-only, forward-only recordset. Using the ExecuteReader method and a SqlDataReader object is the fastest way to retrieve data from a data source, and should be used if the data isn't going to be updated (e.g. it is simply being displayed, and nothing more). As you can see from figure 8.2, SqlDataReader isn't a generic data class, and is actually part of the Managed Provider. The OLE DB Managed Provider also include a DataReader class, and all 3rd party ones should as well, to allow fast access to data.

Finally, the ExecuteNonQuery method is used when you are executing a query that will not return any data, such as an UPDATE or INSERT query. ExecuteNonQuery will return an Integer indicating the number of rows that were affected by the query.

As you can see, the essence of the Command object from ADO remains in the ADO.NET implementation, but it's been given a fair deal more new functionality to better cope with the different types of SQL queries, and XML support should you need it.

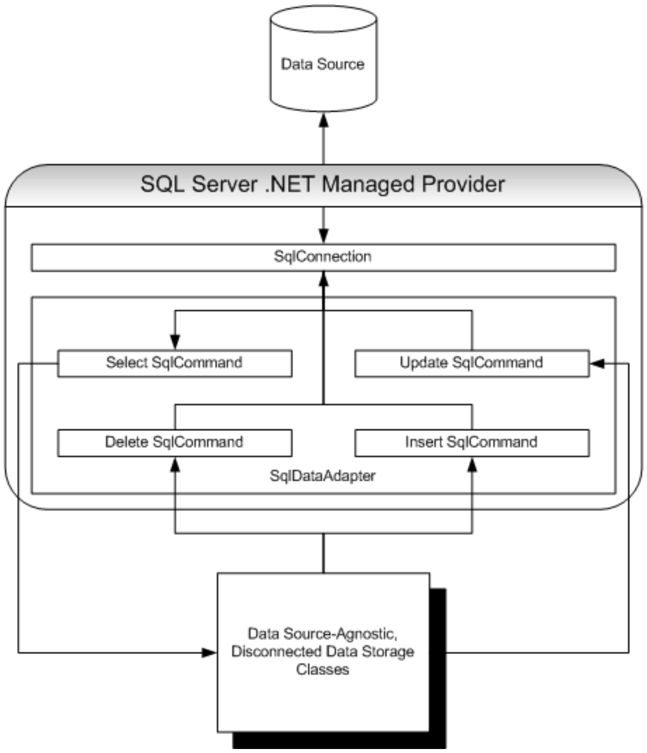
The SqlDataReader Class

As was mentioned in the previous section, the SqlDataReader is roughly equivalent to a read-only, forward-only ADO RecordSet object. One of the annoyances of the ADO RecordSet was that in order to progress to the next record, you needed to call the object's MoveNext method, and apparently many developers often forgot to do this when they were looping through a RecordSet. I know I'm certainly guilty of this, and have on many occasions realized my mistake only after loading the offending page and staring at Internet Explorer's loading status into infinity, causing me to have to restart IIS. Because of this, Microsoft designed the SqlDataReader so that you don't have to call a MoveNext method to progress to the next record, and there isn't an EOF property which returns whether you're at the end of the result set or not. In fact, these two functions have been merged into one method, Read, which attempts to proceed to the next record and returns a Boolean – True if it could proceed to the next record, and False if it has reached the last record and obviously could not proceed. There are numerous ways of retrieving data from a SqlDataReader, but possibly the easiest is by calling the Read method, and then using the Item property, which is an indexed property that allows you to pass either the index of the column you want to retrieve data from in the current row, or alternatively you can pass the column name as a string. This works very similarly to the way that data is retrieved from an ADO RecordSet.

The SqlDataAdapter Class

The SqlDataAdapter class has no real ADO equivalent, but it's actually very easy to understand, just so long as you don't try and relate it to anything from ADO. Essentially a DataAdapter is responsible for both retrieving (i.e. SELECT statements) and manipulating (INSERT, UPDATE and DELETE statements) data in the disconnected data classes. It is the "link" if you will, between the Managed Provider and the disconnected data. The SqlDataAdapter has 4 properties that define the SqlCommand objects that control these 4 operations – SelectCommand, InsertCommand, UpdateCommand and DeleteCommand. As their names suggest, they each represent a SqlCommand object that contains a query for retrieving records, as well as inserting, updating and deleting records respectively. The DataAdapter controls when and how these SqlCommand objects are executed, so once you've assigned them, you don't need to do anything more with them. Although it may seem fairly complex, the DataAdapter is very easy to use, and beyond setting the appropriate SqlCommand objects (and if you don't plan on manipulating the data, you only need to set a SqlCommand), you don't have to do much for it to work. The only two methods you need to know are Fill and Update. Fill, as its name suggests, fills a disconnected data storage object (passed as an argument) with the data obtained by executing the SelectCommand SqlCommand. Update, also as its name suggests, executes the necessary INSERT, UPDATE and DELETE queries through the respective SqlCommand objects to update the data source such that it is the same as the data in the disconnected storage object. It does this by working out which rows have changed in the storage object (which incidentally is also passed as an argument), and then executing the appropriate SqlCommand depending on whether a row was updated, inserted or deleted. Figure 8.3 shows the structure of the SqlDataAdapter class.

Figure 8.3 – The SqlDataAdapter Architecture



In essence, the purpose of the DataAdapter is to automate the process of filling the disconnected storage and updating the data store when the contents of the storage change. It reduces a large amount of "plumbing" that would have to be done if it didn't exist, because you'd have to fill the storage classes manually, manually iterate through them checking for changed rows and manually execute the appropriate SQL query for each changed row.

We've now covered the basic functionality offered by the 'essential' Managed Provider classes, so it's time to move on to the generic, database-agnostic data storage classes.

The Disconnected Data Storage Classes

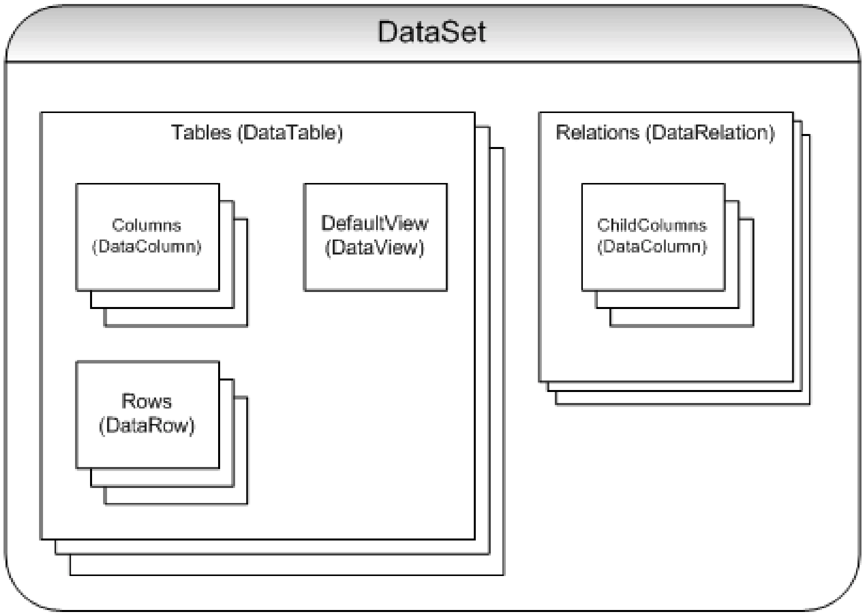
Just as the Managed Providers introduce several new classes that don't have ADO equivalents, so too do the disconnected classes. In fact, trying to make comparisons between these classes and anything in ADO will probably just confuse you, because you'll then expect them to operate in ways that they weren't designed to. Just so that you've got this clear in your own mind from the beginning, there is no RecordSet class in ADO.NET. That's right, there is no RecordSet class; so don't read this section trying to compare each class I give an overview of in terms of one, because although the functionality of the RecordSet is implemented in pieces by the classes, there is no one class that behaves anything like it.

> Many .NET trainers and authors have compared the .NET DataSet (you'll discover what a DataSet is in a moment) to a RecordSet, saying that it is the closest thing in .NET to a RecordSet, but I disagree. To me, the SqlDataReader seems must closer to a RecordSet than any other .NET class, simply because it performs the same function as a RecordSet (or at least part of the function of a RecordSet), and it operates in a very similar way. I'll leave the final judgment up to you.

The DataSet Class

Since the DataSet doesn't apply specifically to any one Managed Provider, it is simply included in the System.Data namespace. The DataSet is the central class for storing disconnected data – it is the "glue" that holds everything to do with the data together, and contains a set of collections of other objects, which hold the data and information about it. Figure 8.4 shows how the DataSet is structured.

Figure 8.4 – The DataSet Class



In the diagram, the property names are shown followed by the their type in parentheses. There are only two properties of the DataSet that you need to know at this point – Tables and Relations. As you can see, the Table property stores a collection of DataTable objects, and the Relations property stores a collection of DataRelation objects. The DataSet, unlike any object in ADO, can contain multiple tables of data, and thereby multiple sets of results from a database. Although this in itself is convenient, it is made even more powerful by the set of relations that a DataSet stores – these can be used to define how tables in a DataSet relate to each other, and is designed specifically for parent-child relationships, such as a Customers table and an Orders table. However, we'll begin with the class that stores the data in a DataSet, the DataTable.

The DataTable Class

The DataTable class, as the name suggests represents a single table of data in memory. However, this table need not map directly to a table in a database, since the DataSet is separated from the data source, and can therefore store fields from multiple different tables in a data source in a single DataTable if need be. Typically the DataAdapter will be used to fill a DataTable object inside a DataSet, and this can be manipulated later through the Tables property. However, a DataTable doesn't have to be created by the DataAdapter, and you can also create your own DataTable objects manually and add them to a DataSet.

The DataTable has several important properties that are essential for manipulating the data contained within. The TableName property, which is a string, is used to uniquely identify a table within a DataSet, and the DataSet's Tables index property allows either the index of a table, or a table's name to be passed, and it will return a reference to the respective DataTable object.

The Columns property is a reference to a DataColumnsCollection object, which is a collection of DataColumn objects. It allows columns to be added, edited or removed through its Add and Remove methods, and its Item property, which allows a specific DataColumn object to be manipulated. You will not normally use the Columns property a great deal if you are loading data from a DataAdapter, as it creates the appropriate DataColumn objects automatically, but if you are creating a DataTable manually you will have to add columns to the table.

The DataColumn Class

The DataColumn provides a fairly comprehensive means of defining columns in a table. Its ColumnName, DataType and DefaultValue properties allow the column name, the type of data that the column holds and the default value for new rows if the column value is not explicitly set to be specified. In addition, the AutoIncrement property (and its accompanying AutoIncrementSeed and AutoIncrementStep properties) allows the creation of auto incrementing columns. The Unique property can also be used to ensure that a column has a unique value.

The ColumnName of a DataColumn is used to retrieve values from that column in the DataRow class, because DataColumn objects are simply column definitions and don't hold any data themselves. When the DataAdapter is used to create a DataTable in a DataSet, it creates the DataColumn objects using the field names from the Select SQL query that is used to retrieve the data. For example, if the following query were executed, then the DataColumn objects would have the ColumnName property values CustomerName, ContactName and ContactTitle.

```
SELECT CustomerName, ContactName, ContactTitle FROM Customers
```

The DataAdapter will also automatically detect the data type and other information about the columns mentioned in the SQL query and will set the appropriate values in each DataColumn object.

The DataRow Class

The Rows property, like the Columns property of the DataTable, is a collection of objects, namely a DataRowsCollection. This contains a collection of DataRow objects, and the DataRow is a class that you will use continually when dealing with data access. It is possible to enumerate through all the DataRow items in the collection, or you can use the DataRowsCollection Find method, which will return a single DataRow where the primary key value of the row matches the one passed as an Object argument in the Find method. The Item index property will return the single DataRow at the specified index position in the collection.

The DataRow represents a single row of data in a DataTable, and exposes an Item index property that allows the values from each of the cells (columns) that make up the row to be accessed. The values can be retrieved, as well as edited through the Item property, which is of type Object, and can have either the index of the column or the name of the column (as defined by the corresponding DataColumn object in the DataTable’s Columns collection) passed to it to identify which cell should be accessed.

The DataRow contains another interesting property, RowState, which you will rarely access directly but is useful to know about nonetheless. The RowState property is of type DataRowState, an enumeration that describes the row’s state in the DataTable, such as ‘Unchanged’, ‘Modified’ or ‘Added’, for example. It is a read-only property and is set automatically by the system. When the DataAdapter is attempting to determine if and how a row has changed to see what SQL query should be executed when its Update method is called, it uses each DataRow object’s RowState property. When the DataAdapter fills a DataTable, all the DataRows’ RowState properties are set to the Unchanged enumeration variable, but if you modify a row, its RowState flag with change. Similarly, if you add a row, it will be marked as Added, and if you remove a row, it will be marked as ‘Deleted’. You can actually reset the RowState to Unchanged for all the rows in a DataTable by calling the DataTable’s AcceptChanges method, although if you call a DataAdapter Update, this is done automatically.

The DataView Class

When displaying data, you will probably be binding the data from a DataTable (in a DataSet) to a data-displaying Web Control, for example, a DropDownList. However, you may wish to display the data in the DataTable in a different way, or you may only want to display a small portion of it. This is where the DataView class fits in. Essentially all a DataView provides is a “front” for a DataTable, representing the data in it to data-bindable controls. The DataView allows the data to be both sorted and filtered using its Sort and RowFilter properties, which both accept string values. These provide the power of SQL, but without the need to reconnect to the SQL Server to obtain the desired results.

The DataTable’s DefaultView property is a reference to a DataView object, which by default simply provides a view of all the data’s rows and columns without any addition filtering or sorting. Since data binding is so common, this property is provided more as a convenience than a necessity, so you don’t have to create a separate DataView object every time you want to bind data to controls.

The DataRelation Class

The DataSet allows parent-child relationships between its tables to be defined through the use of the DataSet Relations property. This property is a reference to the DataRelationCollection class, which is a collection of DataRelation objects. A DataRelation is very easy to create – the class constructor accepts the name of the relation as a string, followed by two DataColumn parameters – the first being the primary key column from the parent table, and the second being the foreign key column from the child table. Once the DataRelation has been added to the DataSet’s Relations, it is immediately usable. When accessing a DataRow from the parent DataTable, you can call the DataRow’s GetChildRows method, passing the name of the applicable relation as a string, and the method will return an array of DataRow objects from the child table.

Typed DataSets

Another new concept in ADO.NET is that of “typed DataSets”. Previously in database development, all code would interact with tables, rows and columns without being aware of what type of data was being held. For example, to retrieve the name of a customer from a DataTable holding customers, you would have to reference which table contains the customers, and which column contains the customer’s name. Typed DataSets are an extension of the regular DataSet class (through inheritance) that try to abstract the data so that when dealing with the DataSet, you concern yourself with “Customer” objects, and the “CustomerName” property of a “Customer” object, rather than the “Customer Table” and the “CustomerName column of the table”. The primary advantage of using typed DataSets is that you will receive IntelliSense and AutoComplete functionality from the IDE, because the DataSet class exposes rows and columns as actual properties of the customized DataSet object.

The downside of Typed DataSets is that a new derivative DataSet class must be created for each table of data that you use. Fortunately the Visual Studio.NET IDE provides a generation tool that allows you to specify which table you want a typed DataSet generated for, and it will automatically create the class for you. Although this saves a lot of the time and effort that would be required if you were to manually build a typed DataSet class, it still means that if your data schema changes you will need to recreate the typed DataSet.

Summary

The architecture of ADO.NET is quite a significant departure from that of ADO. The introduction of the new DataSet class, which allows multiple result sets to be stored, and then related to each other using parent-child relationships, provides a very powerful framework for working with disconnected data. Even though the connections to databases may be created and maintained by different Managed Data Provider classes, all reading and manipulation of data can be done using the generic DataSet class in a disconnected fashion. If however you need to simply read and display data, you can avoid the overhead of a DataSet by opting to use the DataReader class, which provides a fast, read-only, forward-only way of reading data from a database. Since all Managed Providers must implement certain interfaces, they will all be very similar to use, and still provide much greater performance than in previous versions of ADO by optimizing connections for the specific data sources that they connect to and communicate with.

Using ADO.NET and Databases in the IDE

Although ADO.NET is part of the BCL, and can therefore be used in any .NET application regardless of whether it is built in Visual Studio.NET or not, the VS.NET IDE provides numerous tools that make working with databases and integrating data into applications with ADO.NET a lot easier. However, it is important to remember that whenever you’re using the IDE to help build data applications that all the IDE is doing is generating ADO.NET code and inserting it into your source files for you – nothing more. Anything that you can do using the IDE tools is possible by creating ADO.NET code manually, if you so wish.

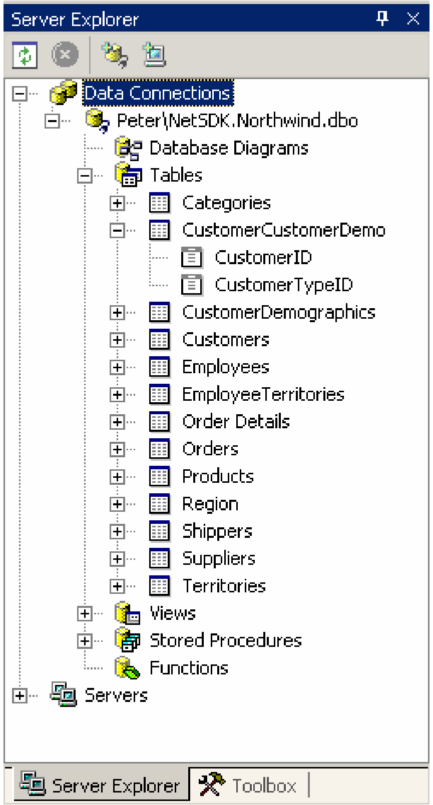
Visual Studio.NET Data Tools

The Visual Studio.NET IDE truly is an integrated environment for building data applications. It includes a multitude of tools and pieces of functionality that make it possible to perform most database management tasks straight from within the IDE – from connecting to database servers, to creating new databases, to view, editing and modifying database items such as table, views and stored procedures. Unfortunately much of this functionality is limited to SQL Server databases, and most other databases are treated as 'read only' where their data can be viewed and modified, but it is not possible to edit the design of tables, stored procedures and so forth. If you are using SQL Server though, you will find these built-in tools to be very useful and won't find much reason to use Enterprise Manager (SQL Server's primary administration tool) at all. All these tools are entirely separate from ADO.NET, and have nothing to do with it except that both ADO.NET and the database tools help build data-aware applications.

All the data management tools are based in the Server Explorer window. There are two ways to manage data using the Server Explorer – the first is through the Data Connections node, which can connect to any OLE DB compatible database, and the SQL Servers node of a particular server in the Servers node. However, when connecting to a SQL Server database, both methods allow you to manage the same aspects of the database. In addition, the Data Connections in the Server Explorer are provided as options to connect to when creating ADO.NET objects in the IDE, so we'll focus on using the Data Connections node for management.

Figure 8.5 shows the Server Explorer after a connection has been made to the Northwind sample database that is shipped with both SQL Server and Microsoft Access.

Figure 8.5 – The Server Explorer, managing the Northwind database on a SQL Server

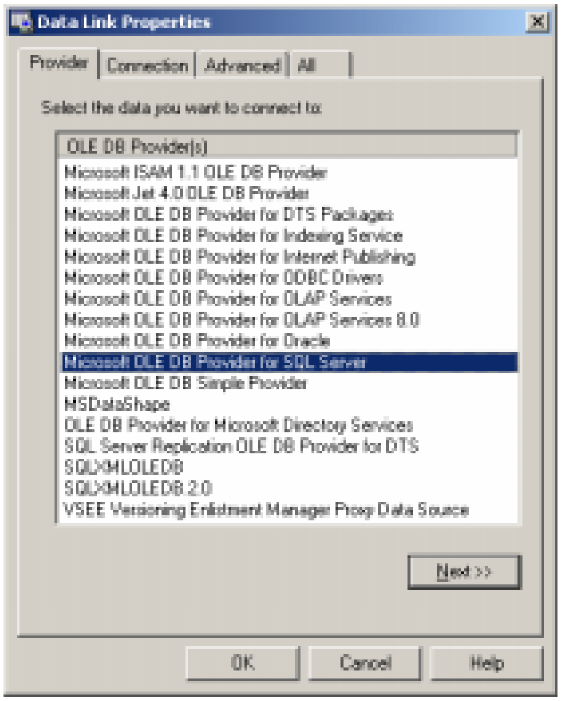


As you can see, all of the important aspects of the database are shown, such as Database Diagrams, Tables, Views, Stored Procedures and User Functions. Before we go over what each node can do, we’ll go through the process of adding a new Data Connection.

Adding a new Data Connection

Before you can access and modify a database, you need to create a connection to it. This is a fairly simple process, which you start off by right clicking the “Data Connections” node and choosing “Add Connection…” The “Data Link Properties” dialog should appear, and will be showing the second tab. Switch to the first tab, “Provider”, and the dialog should look like figure 8.6.

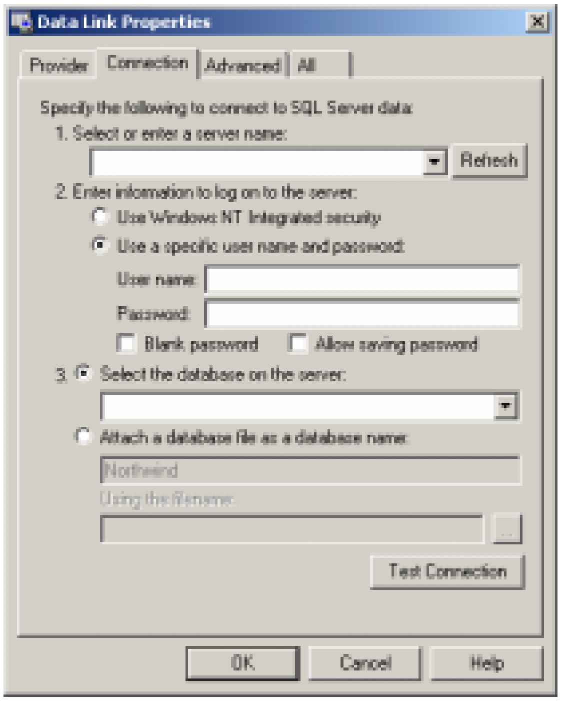
Figure 8.6 – Choosing an OLE DB Provider for a new database connection


Data Link Properties
Provider
Connection
Advanced
All
Select the data you want to connect to:
OLE DB Provider(s)
Microsoft ISAM 1.1 OLE DB Provider
Microsoft Jet 4.0 OLE DB Provider
Microsoft OLE DB Provider for DTS Packages
Microsoft OLE DB Provider for Indexing Service
Microsoft OLE DB Provider for Internet Publishing
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for OLAP Services
Microsoft OLE DB Provider for OLAP Services 8.0
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider
MSDataShape
OLE DB Provider for Microsoft Directory Services
SQL Server Replication OLE DB Provider for DTS
SQLXMLOLEDB
SQLXMLOLEDB 2.0
VSEE Versioning Enlistment Manager Proxy Data Source
Next >>
OK
Cancel
Help

By default the OLE DB provider for SQL Server databases is selected, but as you can see there are numerous other options for other database types. The other most notable ones are the Jet provider, which allows access to Jet data sources such as Microsoft Access, the Oracle provider for access to Oracle databases and the provider for ODBC compatible databases. This allows for access to all ODBC data sources that don’t have OLE DB drivers. Of course, if your installation doesn’t include OLE DB drivers for a particular type of database that you need to use, you should be able to download them from the database vendor – even the open source database MySQL has ODBC and OLE DB drivers available.

Once you’ve selected the appropriate Provider, you can click the “Next >>” button. The options displayed in the “Connection” tab, which should now be shown, will vary depending on which OLE DB provider you’ve chosen. For a Jet source, you’re simply asked for a filename and login details, but the options for a SQL Server database, as shown in figure 8.7, are slightly more involved.

Figure 8.7 – Connection options for connecting to a SQL Server through OLE DB



Specify the following to connect to SQL Server data:

1. Select or enter a server name:

2. Enter information to log on to the server:

○ Use Windows NT Integrated security

● Use a specific user name and password:

User name:

Password:

□ Blank password □ Allow saving password

3. ● Select the database on the server:

○ Attach a database file as a database name:

Northwind

Using the filename:

The first step is to enter in the server name you want to connect to. Next, you choose which authentication method you'd like to use to connect to the server, and if applicable, enter your login details. The final step is to choose which database on the server you will be working on. You can then ensure that the connection is working by clicking the "Test Connection" button, which should display a confirmation message if the test succeeded, in which case you can click "Ok" to add the new Data Connection. The extra node should appear in the Server Explorer, and depending on the functionality offered by the database and VS.NET, you'll be able to modify the following database components.

Database Diagrams

The Database Diagrams node expands (if the database contains any diagrams) to show the name of each diagram, which can then be viewed, edited or deleted through the right-click context menu. Additionally, right-clicking the node and choosing the "New Diagram" option can be used to create new database diagrams. The diagram editor is fully integrated into the IDE, and displays as another tab in the main editing environment. It looks and acts identically to the diagram editor in SQL Server's Enterprise Manager.

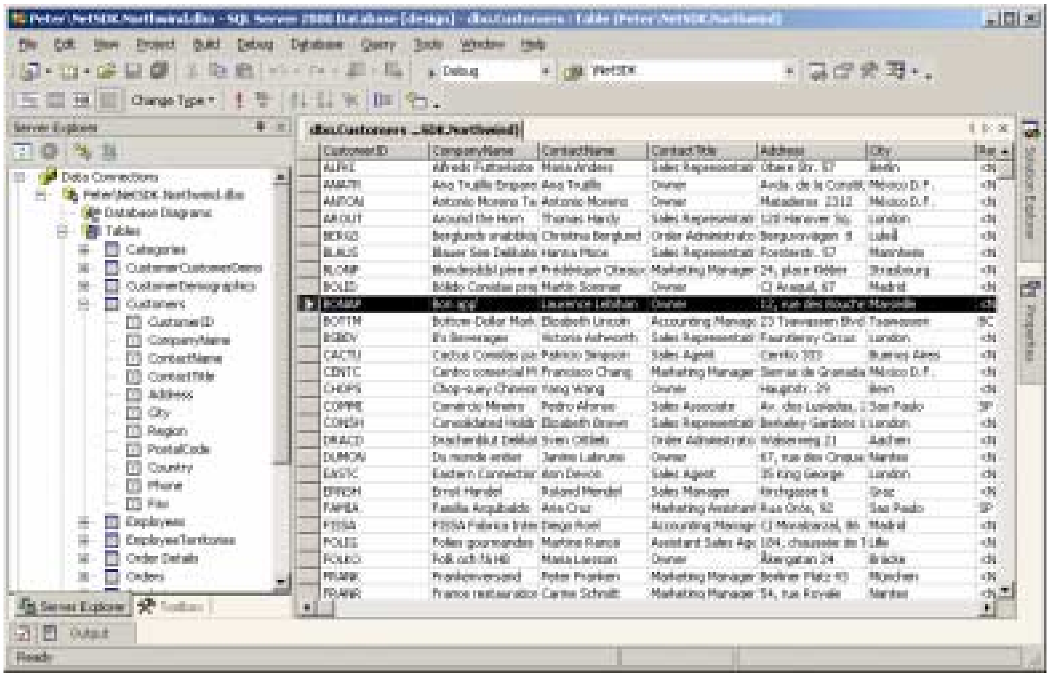
Tables

The Tables node displays a list of the tables contained in the database, and each column in a table is displayed as a child node of the table. Properties about the columns, such as maximum length and data type can be obtained by selecting the column node and looking at the Properties Window.

VS.NET provides extensive functionality for managing tables. You can retrieve and edit data in a table, as well as modify the design of a table from within the IDE. It is also possible to add triggers, as well as entire new tables. Rounding off the functionality is the ability to export table data, and drop (remove) tables.

To access the data editor for a table, right click the table node and choose "Retrieve Data from Table" from the context menu. The editor should be displayed in the IDE's editing area, similar to figure 8.8.

Figure 8.8– The VS.NET Table Data Editor



CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Reg
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representati	Obere Str. 57	Berlin	<N
ANATR	Ana Trujillo Empare	Ana Trujillo	Owner	Avda. de la Constit	México D.F.	<N
ANTON	Antonio Moreno Ta	Antonio Moreno	Owner	Mataderos 2312	México D.F.	<N
AROUT	Around the Horn	Thomas Hardy	Sales Representati	120 Hanover Sq.	London	<N
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrato	Berguvsvägen 8	Luleå	<N
BLAUS	Blauer See Delikate	Hanna Moos	Sales Representati	Forsterstr. 57	Mannheim	<N
BLONP	Blondesddsl père et	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg	<N
BOLID	Bólido Comidas prep	Martín Sommer	Owner	C/ Araquil, 67	Madrid	<N
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouche	Marseille	<N
BOTTM	Bottom-Dollar Mark	Elizabeth Lincoln	Accounting Manage	23 Tsawassen Blvd.	Tsawassen	BC
BSBEV	B's Beverages	Victoria Ashworth	Sales Representati	Fauntleroy Circus	London	<N
CACTU	Cactus Comidas par	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Aires	<N
CENTC	Centro comercial M	Francisco Chang	Marketing Manager	Sierras de Granada	México D.F.	<N
CHOPS	Chop-suey Chinese	Yang Wang	Owner	Hauptstr. 29	Bern	<N
COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate	Av. dos Lusíadas, 2	Sao Paulo	SP
CONSH	Consolidated Holdi	Elizabeth Brown	Sales Representati	Berkeley Gardens 1	London	<N
DRACD	Drachenblut Delikat	Sven Ottlieb	Order Administrato	Walserweg 21	Aachen	<N
DUMON	Du monde entier	Janine Labrune	Owner	67, rue des Cinqua	Nantes	<N
EASTC	Eastern Connection	Ann Devon	Sales Agent	35 King George	London	<N
ERNSH	Ernst Handel	Roland Mendel	Sales Manager	Kirchgasse 6	Graz	<N
FAMIA	Familia Arquibaldo	Aria Cruz	Marketing Assistant	Rua Orós, 92	Sao Paulo	SP
FISSA	FISSA Fabrica Inter	Diego Roel	Accounting Manage	C/ Moralzarzal, 86	Madrid	<N
FOLIG	Folies gourmandes	Martine Rancé	Assistant Sales Ag	184, chaussée de T	Lille	<N
FOLKO	Folk och fä HB	Maria Larsson	Owner	Åkergatan 24	Bräcke	<N
FRANK	Frankenversand	Peter Franken	Marketing Manager	Berliner Platz 43	München	<N
FRANR	France restauratio	Carine Schmitt	Marketing Manager	54, rue Royale	Nantes	<N

This functionality is available for most databases, and is very useful when you quickly need to check whether you code has indeed added, modified or removed table entries as it was supposed to.

The IDE also includes an integrated table designer, so you can both modify existing table designs or create new tables. The designer is identical to the one in the Enterprise Manager, so there's no longer any need to leave the VS.NET IDE if you want to add or edit tables. Both these options are accessible via the right-click context menu of a table node, as well as an option to drop the table (which is quite useful if your code creates temporary tables, but crashes before it gets to the cleanup code where the tables are dropped).

The final significant options in the table node context menu are those to add a trigger to the table; generate the SQL code that creates the table and export the table data to a file. Adding a trigger results in the VS.NET IDE SQL code editor being displayed, which provides a great environment for writing SQL code. We'll look at it in more detail in the 'Stored Procedures' section. The other two options are fairly self-explanatory, and might be useful when you're deploying the application.

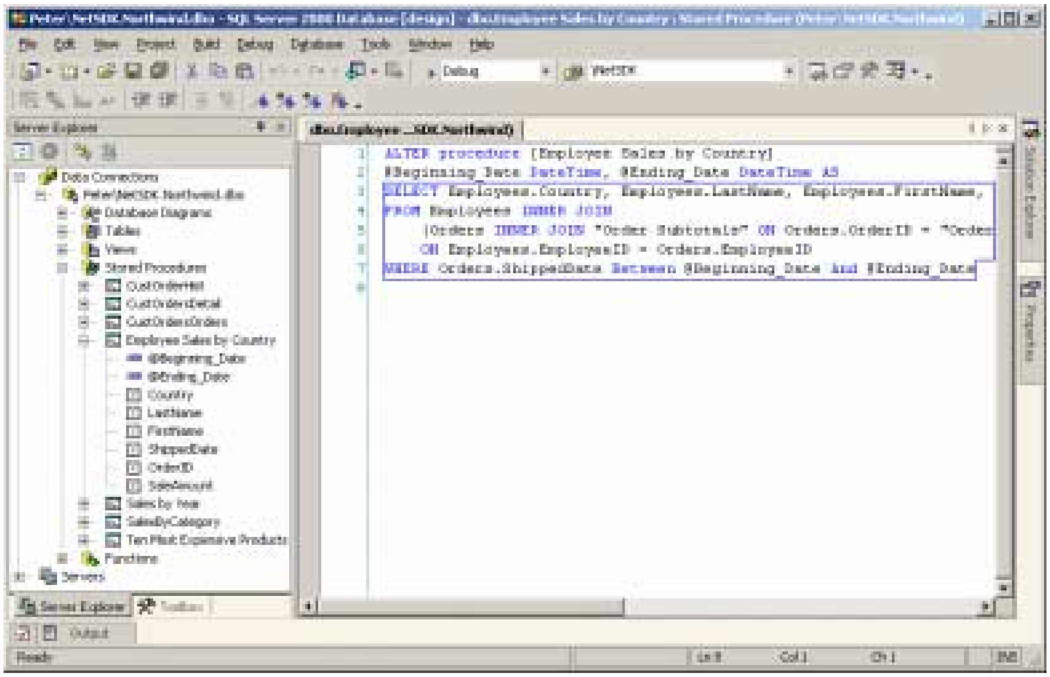
Views

Views provide the same options as for tables, except for extracting the data – namely the ability to review the data in an editing window, as well as design the views and add triggers. All the columns involved in a View are shown as child nodes of the view node, and can be inspected using the Properties Window.

Stored Procedures

Because stored procedures in SQL Server are compiled and thereby offer greater performance than executing simply SQL text queries, you should try to use stored procedures wherever possible if the query is going to be used in a high-load page. Stored procedures also alleviate many problems associated with thwarting “SQL insertion” security breaches. Because of this, you’ll probably spend a fair amount of time building and revising stored procedures, which incidentally are also a great place to store business logic. Visual Studio.NET provides a very useful stored procedure editor, which I would venture to say is much better than the Enterprise Manager’s offering. To edit a stored procedure, right click it and choose “Edit Stored Procedure” from the context menu, or you can add a stored procedure by choosing the appropriate option from the menu. The editor includes the obligatory code syntax highlighting functionality, but the most useful feature is the “Query Builder”. Any INSERT, UPDATE, DELETE or SELECT SQL block is outlined with a blue border. This indicates that the Query Builder can be used to help construct the statement by right clicking anywhere inside it, and choosing the “Design SQL Block” option. Figure 8.9 shows the stored procedure editor with a SQL statement that can be edited using the Query Builder.

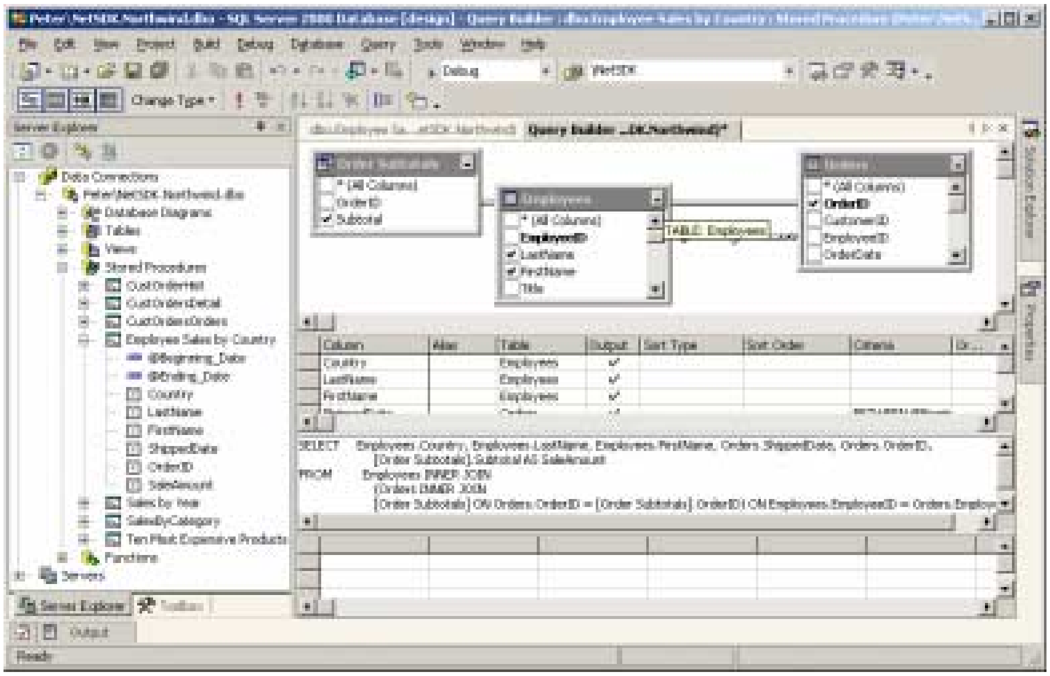
Figure 8.9 – The Stored Procedure Editor



```
ALTER procedure [Employee Sales by Country]
@Beginning_Date DateTime, @Ending_Date DateTime AS
SELECT Employees.Country, Employees.LastName, Employees.FirstName,
FROM Employees INNER JOIN
    (Orders INNER JOIN "Order Subtotals" ON Orders.OrderID = "Order
   ON Employees.EmployeeID = Orders.EmployeeID
WHERE Orders.ShippedDate Between @Beginning_Date And @Ending_Date
```

The Query Builder is very similar to its equivalent in Enterprise Manager, except it is now available when editing stored procedures, which it isn’t in SQL Server’s default administration tool. This really is a time-saver, especially when you’re building complex queries involving multiple table joins, for example. The Query Builder has four main panes, as shown in Figure 8.10. The first is the “Tables” pane, which shows the tables that the query involves, how they relate to each other (foreign key relationships etc) and what columns are available in each. Extra tables can be added to the query by right clicking in the blank space in the pane and choosing “Add Table”. Once you have the tables you need, you select the columns from each table using the checkboxes. These selections will be shown in the second pane, which allows you to specify whether the column will be outputted (in the case of a SELECT statement), whether it should be the basis for sorting the results and whether it should form part of the criteria as to which rows are affected (the WHERE clause). The third plane shows the generated SQL code, which you can modify and the changes will be reflected accordingly in the previous two panes, and the final pane shows the results of the query if you run it by right clicking anywhere in the Query Builder and choosing “Run”.

Figure 8.10 – The Query Builder



Column	Alias	Table	Output	Sort Type	Sort Order	Criteria	Or...
Country		Employees	✓				
LastName		Employees	✓				
FirstName		Employees	✓				

```
SELECT    Employees.Country, Employees.LastName, Employees.FirstName, Orders.ShippedDate, Orders.OrderID,
               [Order Subtotals].Subtotal AS SaleAmount
FROM      Employees INNER JOIN
               (Orders INNER JOIN
               [Order Subtotals] ON Orders.OrderID = [Order Subtotals].OrderID) ON Employees.EmployeeID = Orders.Employ
```

Besides being able to create and edit stored procedures, the node in the Server Explorer also allows you to execute a procedure, but more importantly, “step into” a procedure to debug it, step by step. Both of these options are available through the context menu. Finally, the child nodes of each procedure display the parameters that the procedure receives and outputs, as well as the column names that it will output if it contains a SELECT statement. You can see more about each node by selecting it and viewing it in the Properties Window, which will show details such as data type and length.

Functions

This node displays the list of user-defined functions, and allows you to create, edit or remove them from the database. The same SQL code editor is used when editing a function as for triggers and stored procedures, along with the Query Builder.

Summary

The data tools in Visual Studio.NET help to make VS.NET the only application you need running when building web-based applications. If you're using a SQL Server backend, the functionality provided by the data tools in the IDE is for the most part all you need. There are a few pieces of Enterprise Manager "missing" in the Server Explorer, but all the most common tasks are catered for, and the stored procedure editor is even better than the one included in Enterprise Manager. However, the SQL Query Analyzer and SQL Profiler tools are largely unimplemented in VS.NET, so you'll still need to use them separately.

Building Simple Data Forms

Now that we've gone through the ADO.NET theory and the tools available in VS.NET for managing your databases, we'll move onto actually building Web Forms that access and manipulate data, particularly from the Northwind sample database (included with both Microsoft SQL Server and Microsoft Access). We'll start off with an example that involves a very small amount of code, largely due to VS.NET generating most of it for us. The Visual Basic programmers will be greatly familiar with data binding and data bound controls, so our first example should seem quite familiar to what you're used to. However, for those of you who haven't used VB before, an explanation of what exactly data binding is might be in order.

What is Data Binding?

Data binding was introduced to ASP.NET to help reduce the number of lines of code it requires to display data on a form. Without data binding, it would be necessary to manually write the code to assign values to controls (normally DropDownList, CheckBoxList, RadioButtonList or other more advanced data display controls, such as the DataList and DataGrid) from the database. For example, if you wanted to display the name of each customer in the Customers table in the Northwind sample database, you would have to manually loop through a result set of entries from the Customers table, and write the code to add the entries for each iteration. However, with data binding, you can simply "bind" the results of a query (either in a DataSet or in a DataReader) to a control, and ASP.NET will do whatever is necessary to display the data in your chosen control. This can be a great timesaver, especially when the DataList and DataGrid controls are used, which will be introduced later.

Unfortunately ASP.NET only supports one-way binding – in other words, ASP.NET can set the values of controls according to data from a data source, however, it cannot automatically update the data source when the user changes these controls. Data Binding also has other significant differences to what Visual Basic users will be used to, although the core concept of binding data from a database to a control property still remains. If you're a VB programmer, the first thing you think of when "data binding" is mentioned is probably a form with textboxes and a data control on it, where the user can scroll through the entries one-by-one using the forward and backward buttons on the data control. This is not how data binding works in ASP.NET, and there is no "data control" for you to use, and the TextBox is not a data bindable control – most controls do not provide support for data binding binding, and the TextBox control is not one of them. Table 8.1 shows the list of controls that data can be bound to.

Table 8.1 – Data Bindable Controls

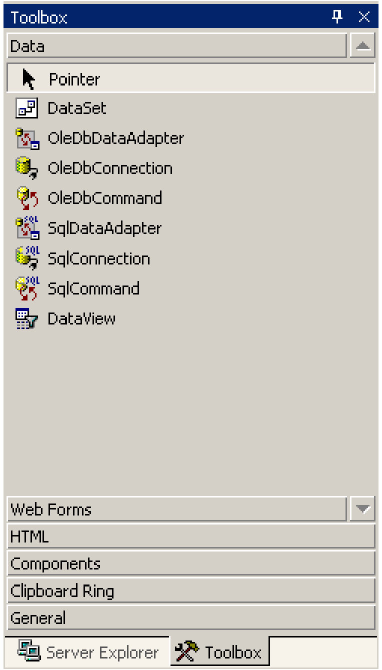
Control Name	**Control Type**
DropDownList	Web Server Control
Dropdown	Html Server Control
ListBox	Web Server Control
Listbox	Html Server Control
DataGrid	Web Server Control
DataList	Web Server Control
Repeater	Web Server Control
CheckBoxList	Web Server Control
RadioButtonList	Web Server Control

Your First Data-enabled Web Form

The first step when building a page that is going to access data from a database is to decide whether to use Visual Studio .NET’s ADO.NET support and create the Connection and other required objects in the IDE, or to code everything manually. There are no major inherent disadvantages of using VS.NET to help speed up development – after all, all VS.NET is doing is writing the same code for you that you would have to write if you didn’t use it. In this section, we’ll build a page that shows off simple data binding, as well as the basics of working with DataSets to retrieve data. We’ll start off by using the VS.NET controls to provide a large part of the required functionality, but I’ll also show you how to build the page from scratch, without any help from the VS.NET IDE for the data access functionality.

So, to begin with, open up a new blank Web Form and then switch to the “Data” tab in the Toolbox, as shown in figure 8.11.

Figure 8.11 – The Data “Controls” in the Toolbox



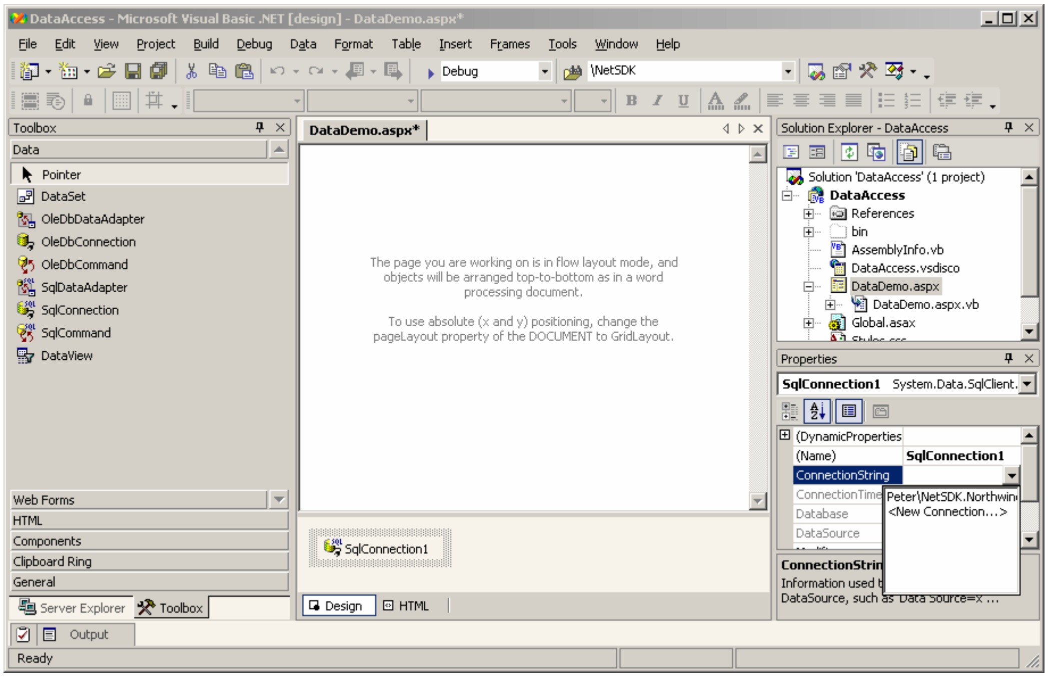
Of course, none of these are "controls" in the conventional sense – they aren't like a TextBox, or a DropDownList. They are "non-visual" controls, because the user doesn't see them on the form as they don't have a UI, for obvious reasons. This means that when you insert any of these controls, no changes are made to the UI file (the HTML). Typically what will happen is that a new object declaration will be added to the code-behind Page class (similar to what happens for regular controls), and any initialization that might be required will be added to the "Web Form Designer Generated Code" region, also in the code-behind file. However, to enable you to modify properties on the ADO.NET objects in the designer, VS.NET will display a separated region at the bottom of the page designer, where all the non-visual controls are represented and can be modified or removed.

Inserting a Connection Object

The first ADO.NET object that you need to insert is a Connection object. As with the previous section, the examples will all use the SQL Server provider, but if you don't have access to a SQL Server machine, you can use the OLE DB provider and connect to a Microsoft Access version of the Northwind database, which is almost identical, except for functionality where stored procedures are required.

Add a SqlConnection (or OleDbConnection, if you need to connect to an Access database) object to the Web Form by dragging and dropping the item from the Toolbox onto the page, just as you would for a TextBox, or any other control. The Designer should create the new section for non-visual controls, and add the SqlConnection object to it, as shown in figure 8.12.

Figure 8.12 – Using Non-Visual Controls in the Designer



The page you are working on is in flow layout mode, and objects will be arranged top-to-bottom as in a word processing document.

To use absolute (x and y) positioning, change the pageLayout property of the DOCUMENT to GridLayout.

Once you've added the Connection, you need to provide a connection string in the ConnectionString property of the SqlConnection object, which defines what data source is going to be used. The connection string comprises of several details, including the location of the data source (in the case of a SQL Server, this would normally be the IP address of the database server, and for an Access database it would be a file path), as well as authentication details and any other details pertinent to the type of database that is being connected to. However, the OLE DB provider requires that one extra parameter be included in the query string, the "Provider", which defines what type of OLE DB provider to use – the SQL Server-specific Connection never requires this, since it will (and can only) always connect to a SQL Server database.

You don't have to manually create the connection string, as VS.NET will do it for you. Simply select the new Connection object and go to the ConnectionString property in the Properties window. If you click on the dropdown arrow (as shown in figure 8.12), a popup will appear showing all the available connections (this list is derived from the Data Connections node in the Server Explorer). You should already have the appropriate connection to the Northwind database created, in which case you can select it, or if not, you can choose the "New Connection" option to set up a new connection to it. After selecting a connection, the ConnectionString property should display text similar to this:

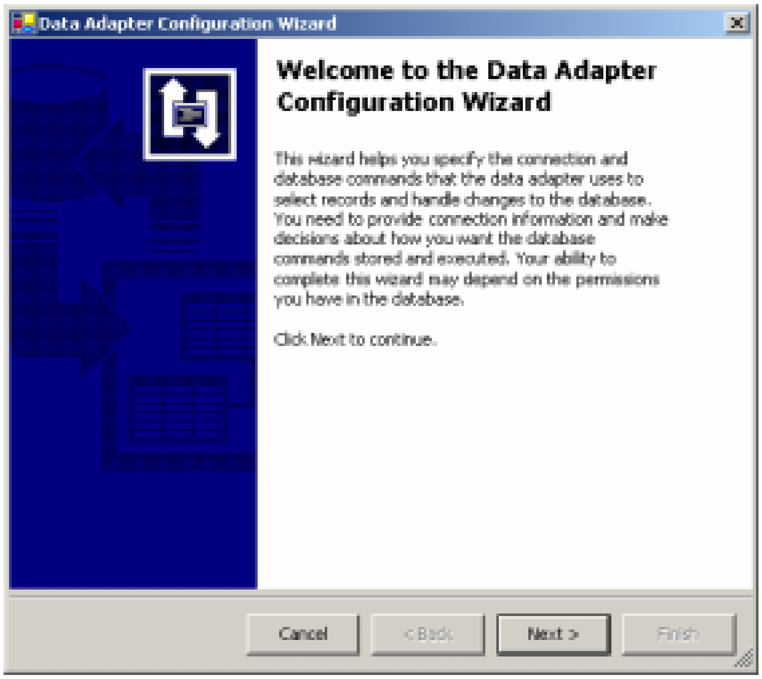
```
data source=Peter\NetSDK;initial catalog=Northwind;persist security
 info=False;user id=sa;workstation id=PETER;packet size=4096
```

As you can see, the different "segments" of the connection string are separated using semi-colons, and the string is constructed using a series of name-value pairs. The three most important pairs in this connection string are "data source", "initial catalog" and "user id". The "data source" specifies the IP address or name of the SQL Server machine, "initial catalog" defines the default database to work with on the database server and "user id" defines the user id to connect to the server with. In this case the server doesn't require a password for SQL Server authentication (although of course a production machine should), but if it did, an additional "password" pair would also be included in the connection string.

Inserting a DataAdapter Object

Once a connection is available, you need a DataAdapter to retrieve data from the database and insert it into a DataSet. Insert a SqlDataAdapter object by dragging and dropping it into the page, and the "Data Adapter Configuration Wizard" should appear, as shown in figure 8.13.

Figure 8.13 – The Data Adapter Configuration Wizard



Welcome to the Data Adapter Configuration Wizard

This wizard helps you specify the connection and database commands that the data adapter uses to select records and handle changes to the database. You need to provide connection information and make decisions about how you want the database commands stored and executed. Your ability to complete this wizard may depend on the permissions you have in the database.

Click Next to continue.

This Wizard makes setting up a DataAdapter significantly easier, as there are a fairly large number of properties that need to be set in a DataAdapter. When configuring a DataAdapter, there are two aspects that you must deal with – firstly, you need to define which connection is going to be used for accessing the database, and secondly, you need to create and set the SQL queries for retrieving records, as well as updating, deleting and inserting records.

After the welcome screen, the first step in the wizard allows you to choose which database connection the DataAdapter must use. If you select a data connection that a Connection object already exists for (such as in the current scenario, when you've already created a SqlConnection object, which provides a connection to the Northwind database), then the DataAdapter will be set up to use that connection. However, if you select a database connection that doesn't already have a Connection object connecting to it, it will automatically create one for you. You should only have one choice in the dropdown, the Northwind database, so select it and move on.

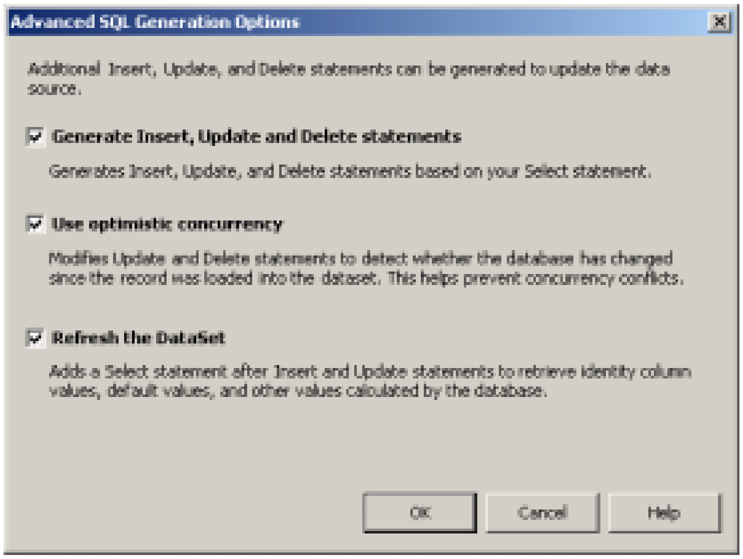
The next step in the wizard allows you to choose how you want the DataAdapter to perform operations on the database server. When using the OleDbDataAdapter and a database that doesn't support stored procedures, only the first option, "Use SQL statements" will be available. However, if you're connecting to a database server that supports stored procedures, you can make the DataAdapter perform all queries (retrievals, updates, insertions and deletions) using stored procedures, which is obviously important, because as you probably know, stored procedures provide far superior performance compared to executing regular SQL statements. The DataAdapter wizard provides extra two options, over and above the option to simply use regular text queries – “Create new stored procedures” compel the wizard to automatically create the appropriate stored procedures for you, and “Use existing stored procedures” allows you to specify the existing stored procedures to be used for the 4 operations that the DataAdapter performs. In this chapter we’ll only deal with using the first option, but using the second is almost identical except that the wizard generates stored procedures, rather than plain SQL statements. The third option will normally be used when the wizard can’t automatically generate stored procedures for you, because your requirements are too complex (this normally happens when trying to use data from multiple tables, but in other scenarios as well). Select the first option and move on.

The third (and final) step in the wizard is to define the SELECT statement for retrieving data. The wizard will (with some exceptions) generate the appropriate INSERT, UPDATE and DELETE statements based on the information given in this SELECT statement if you want it to (if the data you’re retrieving isn’t going to be edited by the user, then you obviously don’t need this functionality). You can either type the SELECT query straight into the text box provided, or use the “Query Builder” tool, which is opened by clicking on the named button at the bottom right of the text box. This will open in a new window, and should look quite familiar – it’s the same tool that is available for editing queries in stored procedures and other database objects from the Server Explorer. Add the “Customers” table for use in the query, and check the CustomerID, CustomerName, ContactName and ContactTitle fields in the window that appears for the table. The query, displayed in the third pane, should look similar to this:

```
SELECT CustomerID, CompanyName, ContactName, ContactTitle FROM
Customers
```

Click “Ok” the close the Query Builder, and return to the wizard. The query should now be displayed in the text box in the wizard. In this example, we’re not going to allow the user to modify the CustomerID or CompanyName fields – we’re simply going to display a list of customers in a DropDownList by binding the data to it, so there’s no need for the wizard to generate the rest of the SQL statements in this case, because they won’t be used. To prevent the wizard from doing this, you need to open the “Advanced Options” dialog (shown in figure 8.14), by clicking the button at the bottom left.

Figure 8.14 – Advanced Options in the DataAdapter Configuaration Wizard



Additional Insert, Update, and Delete statements can be generated to update the data source.

☑ **Generate Insert, Update and Delete statements**

Generates Insert, Update, and Delete statements based on your Select statement.

☑ **Use optimistic concurrency**

Modifies Update and Delete statements to detect whether the database has changed since the record was loaded into the dataset. This helps prevent concurrency conflicts.

☑ **Refresh the DataSet**

Adds a Select statement after Insert and Update statements to retrieve identity column values, default values, and other values calculated by the database.

The first option, “Generate Insert, Update and Delete statements” speaks for itself, really. If it’s checked, the wizard will automatically create these statements for you, based on the SELECT statement you provided, and if not, then it doesn’t. Uncheck this option, because this example only needs to retrieve data. The other two options should be grayed out.

Even though you’re not going to use them now, I’ll explain the other two options as well. “User optimistic concurrency” makes the wizard generate the SQL statements in such a way that they help to avoid concurrency conflicts by ensuring that the record the user is trying to modify or delete hasn’t been changed by another user since the DataSet was last loaded (since the DataSet is disconnected, it is highly possible that its contents will differ from the live data in certain scenarios). This is a fairly useful option, and provides a relatively good generic solution for solving the problem of concurrency, although it obviously won’t be perfect for every situation, and you’ll also have to manually decide what to do when a query is not executed because the database row in question has been changed.

The third option, “Refresh the DataSet”, when checked, makes insertion and update queries always refresh the DataSet by calling the SELECT statement after they are executed. This is particularly important when records are added, since “autonumber” or “identity” fields, which are calculated by the database server, will probably be needed as the primary key field of a table, and any columns that weren’t explicitly set in the SQL statement will revert to the database defaults, which might also need to be used later on.

With the advanced options set, click “Ok” to close the dialog, and then “Next” in the wizard to move to the final confirmation screen, which should display the message, “The data adapter ‘SqlDataAdapter1’ was configured successfully.” You can click “Finish” to close the wizard.

If you ever need to re-run the DataAdapter Configuration Wizard, you can do so by right-clicking the DataAdapter object in the non-visual controls section of the designer, and choosing the option, “Configure Data Adapter…”

Inserting a DataSet

The final “control” we need to add from the Data palette in the Toolbar is a DataSet, which will of course be used to store the data from the database, and will be what the visual controls actually on the page will bind to. As you know, the DataSet is data server-agnostic – it uses XML as its backend, and doesn’t rely on any specific database server or provider, so drag a DataSet from the Toolbar onto the page. A dialog will appear asking whether you want a “Typed dataset” or an “Untyped dataset”. An untyped DataSet is perfectly adequate for now (and in fact, using a typed dataset in this scenario wouldn’t be a major advantage and would only waste resources), so select that option and click “Ok”. A generic DataSet object will be added.

Inserting the Visible Controls

Before you move on to building the actual page, you may wish to rename the three ADO.NET objects to more “friendly” names than their defaults, “SqlConnection1”, “SqlDataAdapter1” and “DataSet1”, by selecting each one and modifying its Name property. However, for the sake of simplicity in this example, we’ll move on without using different names.

The page will perform two functions – firstly, it will display a list of the customers in the Northwind database, and secondly, it will display the name of the contact and his/her title for the selected company. To implement this, you need three controls – a DropDownList, a Button and a Label (all of them Web Server Controls). Add the DropDownList and the Button on the same line, next to each other, and add the Label on the line below them. Name them lstCustomers, btnViewContactDetails and lblContactDetails respectively.

Setting the DropDownList up so that it displays the customer list is fairly easy. Firstly, move down to the DataSource property in the Properties window. When you click on the dropdown arrow, a list of the available data sources that the control can bind to will be shown. This will generally include a list of all the DataSet and DataView objects on the page. Select “DataSet1”. Since the DataSet can hold multiple tables, it is also necessary to specify the specific table that must be used. To do this, modify the DataSource property to read, `DataSet1.Tables("Customers")`. If you were using a typed DataSet, the available tables would be shown in the dropdown, but with a generic DataSet you have to type the appropriate value in.

The next step is to set the DataTextField and DataValueField properties. These define the column that the DropDownList will bind to in the chosen DataSource – DataTextField is the column that will be displayed (i.e. for each row in the DataSet, this column value will be set as the Text property of each DropDownList item), and DataValueField is the column that will be stored in each ListItem’s Value property. Set DataTextField to “CompanyName” and DataValueField to “CustomerID”. You will normally bind the Value to a primary key (in this case, CustomerID), so that each DataItem in the DropDownList can be uniquely identified.

With the DropDownList set up, all that remains is to provide that code that will bind the data from the database to the DropDownList. To do this, switch to the code view and add in the code so that the Page_Load event handler reads like listing 8.1.

Listing 8.1

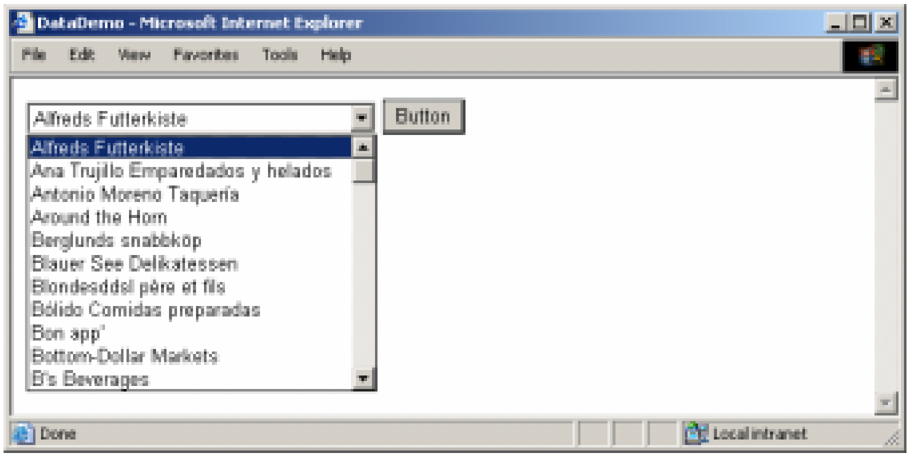
```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
  SqlDataAdapter1.Fill(DataSet1, "Customers")
  If Not Page.IsPostBack Then
    lstCustomers.DataBind()
  End If
End Sub
```

Before you can bind the data, you must first fill the DataSet, otherwise there will be no data to bind to. The SqlDataAdapter's Fill method has numerous overloaded versions, but the one used here accepts two parameters – the first is the DataSet that will be filled, and second is the name of the table inside the DataSet that must be filled. The Fill method executes the DataAdapter's SELECT statement and fills the specified table with the results from the query. If the table does not already exist, it is created. Since the DataSet object is obviously destroyed between postbacks, it must be reloaded every time the page loads, which is why it does not appear in the following If block.

Since the lstCustomers control has already had its data binding properties configured at design time, all that remains to be done is call its DataBind method, which binds the data from its DataSource property to the control. However, the control only needs to be bound once (when the page is first loaded), because ViewState will persist the items in the list through postbacks. The If condition ensures that the DataBind method is only called the first time the page is loaded. If this is not done, then the DropDownList will be reloaded on every postback, and will therefore lose state information, such as the currently selected item.

Although the remaining functionality hasn't yet been implemented, you can load the page, which will show the list of customers in the Northwind database as shown in figure 8.15 – and all it took was 4 lines of code.

Figure 8.15 – Data binding to a DropDownList



Reading data directly from the DataSet

After implementing the binding functionality, the second feature that needs to be built is the code such that when the Button control is clicked, the Label will display the contact person’s name and title for the selected customer. There’s no way to do this using data binding, so we have to manually find the row (record) in the DataSet that corresponds to the selected customer in the DropDownList, and then assign the values of the ContactName and ContactTitle columns in that row to the Label control. Fortunately, because the DataSet provides powerful manipulation features, this is actually fairly simple to do.

There are three steps that need to be performed to complete the task. Firstly, you need to find the index of the selected item in the DropDownList. Secondly, you need to relate the index of the selected item to the indices of the rows held in the DataSet and obtain a reference to the corresponding DataRow object, and thirdly, you need to retrieve the ContactName and ContactTitle column values from that DataRow.

All three of the steps require a line of code each or less. To set up the page appropriately, set the Button’s Text property to “View Contact Details”, and set the Label’s Text to an empty string. You can then double click on the Button control to create a Click event handler for it, and insert the code in listing 8.2.

Listing 8.2

```
Private Sub btnViewContactDetails_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnViewContactDetails.Click
  Dim strContactName As String = _
DataSet1.Tables("Customers").Rows(lstCustomers.SelectedIndex _
).Item("ContactName")
  Dim strContactTitle As String =
DataSet1.Tables("Customers").Rows(lstCustomers.SelectedIndex _
).Item("ContactTitle")
  lblContactDetails.Text = strContactName & ", " & strContactTitle
End Sub
```

These three lines are all that is required. Obtaining the index of the selected item in the DropDownList is very easy, since that control exposes a SelectedIndex property, which returns an Integer value. The DataSet also allows rows to be referenced using indexed values, and since the entire Customers table is bound to the DropDownList, indices from the DropDownList map directly to indices of rows in the DataSet (i.e. if the index of an item in the DropDownList is 5, then the index of the corresponding row in the DataSet will be 5 as well).

As you learned in the ADO.NET theory section, the DataSet stores a reference to a collection of DataTable objects in its Tables property, which is a reference to a DataTableCollection object. This object’s Item property is used to obtain a reference to a specific DataTable object inside the collection by passing either the index of the table (which would be 0 in this case, since no tables were added before the Customers table in the current example), or by passing the name of the table as a string. However, the Item property is the default property, and therefore does not have to be explicitly used, which it isn’t in this code listing. Therefore the following segment of the first line obtains a reference to the “Customers” DataTable object:

```
DataSet1.Tables("Customers")
```

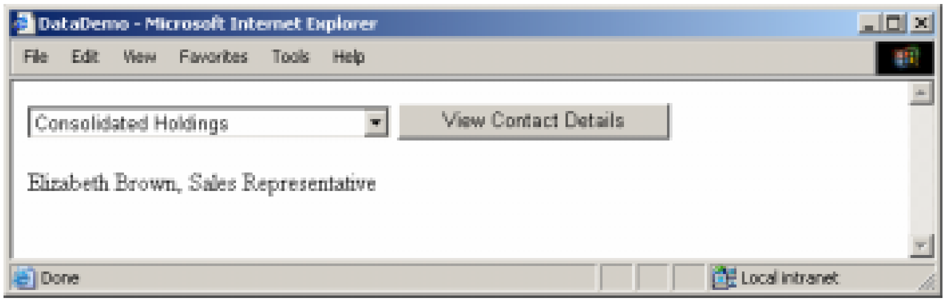
The DataTable’s Rows property is a reference to a DataRowCollection object, which exposes an Item property (which is the default property for the object) that accepts the index of the row to be obtained as a parameter, and returns a reference to the respective DataRow object. Therefore the following segment of the first line obtains a reference to the corresponding DataRow object for the selected item in the DropDownList:

```
DataSet1.Tables("Customers").Rows(lstCustomers.SelectedIndex)
```

With this reference to the appropriate DataRow object, you can obtain the value from any of the row’s columns through the Item property (which is also the default property, but is not used implicitly in the example). The Item property accepts either the index of the column as an integer or the column name as a string, as a parameter, and returns the value of the column. The first line declares a string, strContactName, and using an initializer, sets it to the value of the ContactName column in the selected company’s row. The second line does almost exactly the same, except it declares the variable strContactTitle, which is set to the contents of the ContactTitle column.

With the two values stores in string variables, all that is left to do is display them. The third line simply concatenates these two strings and assigns them to the Text property of the Label so that they are displayed. Figure 8.16 shows the result when the page is loaded.

Figure 8.16 – Retrieving individual values from a DataSet



Consolidated Holdings

Elizabeth Brown, Sales Representative

A Visit Behind the Scenes

As was mentioned at the beginning of this section, the Visual Studio.NET IDE isn’t performing any black magic to data-enable your pages – all it’s doing is writing a fair amount of the ADO.NET code for you that you would otherwise have to write manually. To see this code, open up the “Web Form Designer Generated Code” region in the code-behind file and look at the InitializeComponent method, which is executed before the Page.Load event is fired. It should look similar to listing 8.3.

Listing 8.3

```
Private Sub InitializeComponent()
  Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
  Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
  Me.DataSet1 = New System.Data.DataSet()
  Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
  CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).BeginInit()
  '
  'SqlConnection1
  '
  Me.SqlConnection1.ConnectionString = "data source=Peter\NetSDK;" _
& "initial catalog=Northwind;persist security info=False;user " _
& "id=sa;workstation id=PETER;packet size=4096"
  '
  'SqlDataAdapter1
  '
  Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
  Me.SqlDataAdapter1.TableMappings.AddRange(New _
System.Data.Common.DataTableMapping() {New _
System.Data.Common.DataTableMapping("Table", "Customers", New _
System.Data.Common.DataColumnMapping() {New _
System.Data.Common.DataColumnMapping("CustomerID", "CustomerID"),
New _
System.Data.Common.DataColumnMapping("CompanyName",
"CompanyName"), New _
System.Data.Common.DataColumnMapping("ContactName",
"ContactName"), New _
System.Data.Common.DataColumnMapping("ContactTitle",
"ContactTitle")})})
  '
  'DataSet1
  '
  Me.DataSet1.DataSetName = "NewDataSet"
  Me.DataSet1.Locale = New System.Globalization.CultureInfo("en-ZA")
  '
  'SqlSelectCommand1
  '
  Me.SqlSelectCommand1.CommandText = "SELECT CustomerID,
CompanyName, " _
& "ContactName, ContactTitle FROM Customers"
  Me.SqlSelectCommand1.Connection = Me.SqlConnection1
  CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).EndInit()
End Sub
```

This code makes use of 4 objects, SqlConnection1, SqlDataAdapter1, DataSet1 and SqlSelectCommand1. All four are declared at the top of the class, along with all the other controls on the page (such as the Label). When these objects are declared, they are not created, so in the first four lines of the InitializeComponent method, an instance of each object is created. The remainder of the method is set up into sections where the properties of each object are set and any other initialization is performed. In the first section, for the SqlConnection1 object, only the ConnectionString property is set, which is fairly self-explanatory. The SqlDataAdapter1's SelectCommand property is set in the next section. This property is a reference to an object of type SqlCommand, and is assigned to the SqlSelectCommand1 object, which was created earlier.

The next line might seem quite daunting, but it's actually performing quite a simple operation – when you execute a SQL SELECT statement, the results return column names. However, you may not wish to use the same column names in the DataSet as are returned by the SQL query, which is where "table mapping" enters the picture. Table mapping allows you to define which column the data in each column of a result set must go into in a DataTable object. In this example, we haven't defined any special data table mapping, so the "CustomerID" from the results will be stored in the "CustomerID" column in the DataSet, and so forth.

In the DataSet1 section, another two properties are set – the DataSetName, which also doesn't need explanation, and the Locale property. The Locale property accepts an object of type CultureInfo, which defines the locale used to store data inside the DataSet (specifically, inside the DataTable objects), and is typically used in string comparisons of data.

Finally, the SqlSelectCommand object is set up. Its CommandText property is assigned to the SELECT statement that is later used to fill the DataSet and its Connection property, which defines which SqlConnection object is must use to communicate with the SQL Server, is set to SqlConnection1.

Summary

Now is probably a good time for an overview of what happens in this page, from start to finish. First off, when the page is loaded, the required ADO.NET objects (a SqlConnection, a SqlDataAdapter, a SqlCommand and a DataSet) are all declared and initialized. The SqlConnection is set to use the appropriate SQL Server and database. The SqlCommand is set up to execute a SELECT statement to retrieve a list of customers from the Customers table. The SqlDataAdapter is set to use the created SqlCommand when it needs to retrieve data from the database and fill a DataSet with in. Finally, a DataSet is also needed, so that the SqlDataAdapter can fill it and the rest of the page can retrieve data from it.

With the ADO.NET objects all initialized, the Page_Load event handler is called, where the SqlDataAdapter's Fill method is called. When this happens, the SqlConnection's Open method is implicitly called, and the SqlCommand is executed. The results of the query are placed into a new DataTable in the DataSet, and the SqlConnection's Close method is implicitly called, so that the connection to the database server does not remain open. The remaining code in the Page_Load event handler binds the data in the DataSet to the DropDownList control on the page if the page is being loaded for the first time.

When the Button control is clicked, a postback is triggered, and when the page is loaded, all the ADO.NET objects are again created and initialized, and the DataSet is filled. However, the DropDownList maintains its state, and therefore does not need to be re-bound to the DataSet. The Button control's Click event handler is called, which finds the record in the DataSet that corresponds to the selected customer in the DropDownList, and the ContactName and ContactTitle column values are retrieved and assigned to the Label's Text property. This process is repeated each time the button is clicked.

Building the page by hand

Chapter 9

XML Programming with the XML Base Classes

XML certainly is one of the buzzwords of modern-day software development. Fortunately, unlike most buzzwords, there is actually some substance behind the hype that surrounds XML that make it a very worthwhile addition to your software development toolkit, which is why Microsoft included deeply-rooted support for XML in the .NET Framework. This chapter includes a brief primer on XML, and some of the related “X-” technologies to get you up-to-speed, in case you haven’t yet ventured past the marketing fluff. Following that, the Microsoft XML base classes are introduced, and given a fairly thorough overview, to ensure that you know how to directly manipulate XML data in the .NET Framework. Although the XML support in .NET goes far beyond what is included in the base XML classes, this chapter should arm you with the fundamentals of XML, how to manipulate and use it in .NET, and where it could possibly be useful in your web applications.

Introducing XML

Entire books have been written on XML, so this brief primer certainly doesn’t cover this important topic in any great depth, but it should be sufficient to give you a basic understanding of what XML is all about. XML is an acronym for eXtensible Markup Language. At the most basic level, XML is a format for describing data. However, unlike other database systems such as Access and SQL Server, XML stores data in plain text format. Additionally, XML stores data in a hierarchical format, rather than relational, as used by the more traditional database system, such as SQL Server. The major benefit of using XML over traditional data stores stems from its inherent portability – because XML data is stored in plain text, it is very easy to transport it across platforms, which is why it has been chosen as the format for web services (which are dealt with in Chapter 14, but basically deal with transferring commands and information across websites, irrespective of platform or development framework). However, it is vital to understand that XML is not a replacement for database systems like SQL Server and Oracle, but rather a complementary technology.

A Practical Introduction

XML uses “tags” to describe data, in very much the same way as HTML does. However, although XML data may look like HTML in many respects, the two are completely separate technologies. XML is a generic format for storing (and not displaying) data, whereas HTML is a format that allows data to be presented in a web browser, and uses tags to control the format and layout. In XML, the tags are simply used to delimit that data and provide additional information.

Many aspects of .NET revolve around XML, and it should come as no surprise that Visual Studio.NET also provides support for XML. To create an XML file in a new project, right click on the project node in the Solution Explorer and click Add-Add New Item. In the “Add New Item” dialog, select “XML File”, type in a suitable name, and click Open.

In this section, we’re going to create an XML file that will store a list of employees and related information to demonstrate the basic concepts of XML and establish the meaning of some “XML lingo”, such as XSL, XML Schemas, nodes, attributes and so forth. To start off with, create a new XML file named “employees.xml”. The file will currently only contain one line:

```
<?xml version="1.0" encoding="utf-8" ?>
```

This line is an XML processing instruction (processing instructions are denoted by the question marks at the beginning and end of the tag) that specifies that XML version 1 is being used, and the Unicode character encoding is being used. Now, on to adding actual data. To start off with, we’re going to add the following XML data, so that the document now reads as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<employees>
  <employee>
    Peter McMahon
  </employee>
  <employee>
    Leon Cilliers
  </employee>
  <employee>
    Karl Heydenrych
  </employee>
</employees>
```

In XML, you define your own markup. In other words, you choose which tags you’d like to describe your data, and use them accordingly. Other than for the XML processing instructions, there are no set “tags” that must be used – you invent your own to suite the situation. However, just as HTML documents must conform to certain rules to be considered valid, so too must XML documents. However, web browsers are not very strict in their enforcement of HTML rules, and even invalid HTML documents are displayed as best as the browser can. XML processors are much more stringent in their enforcement of the basic XML structure rules, so it is very important that you make sure that your XML conforms to the basic rules.

One such rule is that all XML documents must contain one, and one only, root “element” or “node”. In XML, an element or node loosely maps to a “tag”, so there is one “employees” element/node and three “employee” elements/nodes in the above example. The root element is the element that contains the rest of the XML data. In this case, the “employees” element is the root element.

Take careful note that for every element, there is a closing tag (i.e. the identical tag, prefixed by a forward-slash). Another XML rule is that all tags must be properly closed. Closing a tag can either take the form of a closing tag, or a trailing forward-slash in the original tag, which will be demonstrated shortly.

Each tag, or element, can contain both text data as well as child nodes. In this example, the root “employees” element contains no text itself, but contains three child nodes, which in turn contain text, but no further child nodes.

Adding Information using Child Nodes and Attributes

For our mini-employees database, we want to contain a little more information about each employee than just his/her name. This can be achieved in two ways – either several child nodes can be added to each employee element to store the data, or each employee element can include attributes that store the additional data. The following sample shows how child nodes can be used to store the additional data:

```
<?xml version="1.0" encoding="utf-8" ?>
<employees>
 <employee>
  <name>Peter McMahon</name>
  <title>Developer</title>
  <salary>60000</salary>
 </employee>
 <employee>
  <name>Leon Cilliers</name>
  <title>Graphic Designer</title>
  <salary>45000</salary>
 </employee>
 <employee>
  <name>Karl Heydenrych</name>
  <title>Project Manager</title>
  <salary>75000</salary>
 </employee>
</employees>
```

In this example, each employee element contains three child nodes storing details about each employee – name, job title and salary. While this method is perfectly acceptable, it would probably be better to include such information in element attributes. XML attributes are identical to their HTML counterparts – they are “properties” that are specified inside the actual tags. The following shows how the above example could be implemented using attributes rather than child nodes:

```
<?xml version="1.0" encoding="utf-8"?>
<employees>
  <employee name="Peter McMahon" title="Developer" salary="60000" />
  <employee name="Leon Cilliers" title="Graphic Designer" salary="45000"
/>
  <employee name="Karl Heydenrych" title="Project Manager"
salary="75000" />
</employees>
```

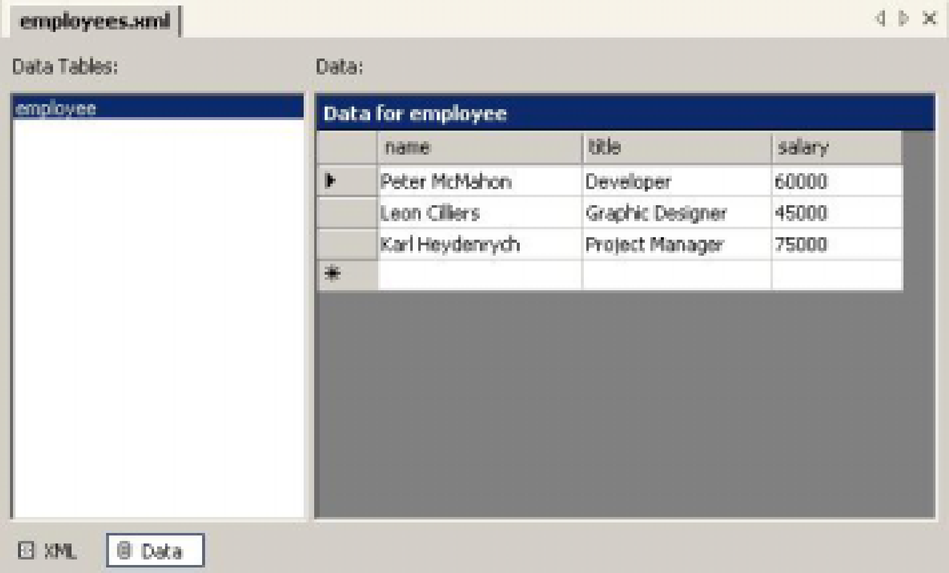
In this example, the trailing forward-slash method of closing each tag is used. This code could also have been written like this, using the closing tag method:

```
<?xml version="1.0" encoding="utf-8"?>
<employees>
  <employee name="Peter McMahon" title="Developer" salary="60000">
  </employee>
  <employee name="Leon Cilliers" title="Graphic Designer"
salary="45000">
  </employee>
  <employee name="Karl Heydenrych" title="Project Manager"
salary="75000">
  </employee>
</employees>
```

The use of attributes is preferable to child nodes when there is a finite number of data that may be applicable to a particular element, whereas child nodes are more appropriate where there is no specific limit to the amount of data that may be related to an element.

Visual Studio's XML Editor's "Data View" mode

As with the HTML designer, the XML designer includes two modes of display – code (the default), and the data view mode. To switch between the two, click on the appropriate button at the bottom left corner of the editing area. For the employees.xml document in its current form, the data display mode will look similar to Figure 9.1.



	name	title	salary
▶	Peter McMahon	Developer	60000
	Leon Cilliers	Graphic Designer	45000
	Karl Heydenrych	Project Manager	75000
*			

[Figure 9.1]

This shows the XML in a "table" format. The "data view" mode of the XML editor in Visual Studio.NET can be a very useful tool when creating and editing XML data stores that structure their data that is in a form that can be represented as a table, although some XML documents can be well-formed, but are still not able to be viewed in the "data view" mode. A good example of this is the web.config application configuration file. However, for data such as our employees information, the “data view” displays the data into a much more readable and usable format.

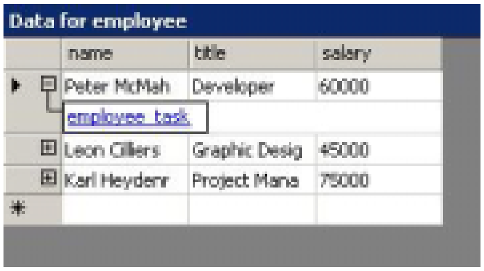
Using both Attributes and Child Nodes

As has already been mentioned, attributes are suitable when an element may have a finite amount of extra data added to it, which in the case of our employees XML file is the employee name, job title and salary. However, if we wanted to add a list of responsibilities, or job tasks, that each employee does, then child nodes would be more appropriate than attributes, since there should be no limit on how many tasks a person is responsible for. For example, a manager might have to task care of client liaison, be responsible for budgeting, prepare reports for upper management, create task schedules for team members and so forth, whilst a graphic designer may only have to do image design and website layout mock-ups. Listing 9.1 is a new rendition of the employees.xml that includes job task/responsibility information as child nodes.

```
<?xml version="1.0" encoding="utf-8"?>
<employees>
 <employee name="Peter McMahon" title="Developer" salary="60000">
  <task>Technical Specification Creation</task>
  <task>ASP.NET and DHTML Programming</task>
  <task>Project Documentation</task>
 </employee>
 <employee name="Leon Cilliers" title="Graphic Designer"
salary="45000">
  <task>Web Image Design</task>
  <task>Website Layout Mock-up Creation</task>
 </employee>
 <employee name="Karl Heydenrych" title="Project Manager"
salary="75000">
  <task>Client Liaison</task>
  <task>Project Budgeting</task>
  <task>Team Task Schedule Creation</task>
  <task>Report Preparation for Upper Management</task>
 </employee>
</employees>
```

[Listing 9.1]

If this is now viewed in the “data view” mode, it is possible to “drill down” into each employee node by clicking on the + next to the item, as shown in Figure 9.2.

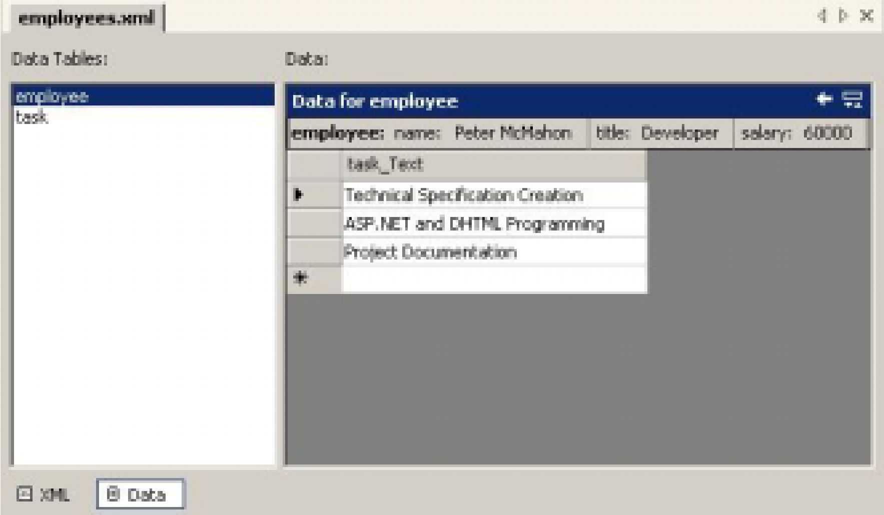


Data for employee

	name	title	salary
▸ ⊟	Peter McMah	Developer	60000
	employee_task		
⊞	Leon Cilliers	Graphic Desig	45000
⊞	Karl Heydenr	Project Mana	75000
*			

[Figure 9.2]

Clicking on the “employee_task” item will result in a display similar to Figure 9.3, which shows all the task nodes beneath the selected employee node.



Data for employee

employee: name: Peter McMahon | title: Developer | salary: 60000

	task_Text
▸	Technical Specification Creation
	ASP.NET and DHTML Programming
	Project Documentation
*	

[Figure 9.3]

Take note that the display not only shows the task nodes, but also information about the selected employee node (in the grey bar directly above the tasks “table”), and in the blue bar above that, clicking the arrow will return you to the table displaying the employee nodes. Take note that the data view has created two separate “tables” for display – “employee” and “task”. Even though XML represents data in a hierarchical format, this demonstrates that it is still possible to manipulate it such that it is displayed in a more relational way.

Adding Additional Information to the Child Nodes

Just as attributes can be added to the employee elements to add additional information about each one, so too can they be added to the child node task elements. It is also possible to mix the combination of elements holding text between their opening and closing tags, and using attributes to hold additional data. Listing 9.2 shows the employees.xml file that stores the priority of each task for the employees in an attribute of the task element, whilst still storing the task description in the element’s text.

```
<?xml version="1.0" encoding="utf-8"?>
<employees>
 <employee name="Peter McMahon" title="Developer" salary="60000">
  <task priority="2">Technical Specification Creation</task>
```

```
    <task priority="1">ASP.NET and DHTML Programming</task>
    <task priority="3">Project Documentation</task>
  </employee>
  <employee name="Leon Cilliers" title="Graphic Designer"
salary="45000">
    <task priority="1">Web Image Design</task>
    <task priority="2">Website Layout Mock-up Creation</task>
  </employee>
  <employee name="Karl Heydenrych" title="Project Manager"
salary="75000">
    <task priority="1">Client Liaison</task>
    <task priority="3">Project Budgeting</task>
    <task priority="2">Team Task Schedule Creation</task>
    <task priority="4">Report Preparation for Upper Management</task>
  </employee>
</employees>
```

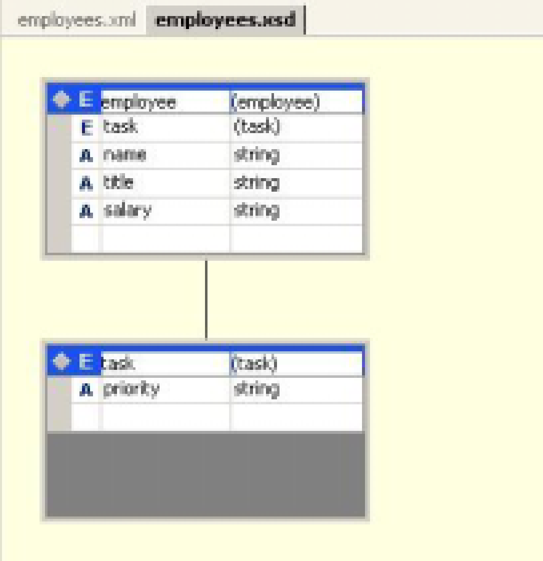
[Listing 9.2]

Inspecting the file in the data view will show that the task “table” entries now have two columns – task_Text and priority.

XML Schemas

In a RDBMS such as SQL Server, you design a database by adding tables, and for each table defining column names, and the data type for each column. Once this design has been set, all entries must conform to it, and there’s no way (other than modifying the actual design) of adding or modifying entries so that they don’t conform to the design, because the RDBMS simply disallows the request. Since XML is doesn’t natively have any rules of what elements can be added to a document, and what attributes and child nodes that element can have, XML can be very unstructured unless an XML schema is used. An XML schema can be thought of as the “design rules” for an XML file; it is almost analogous to the database design in a regular relational database. If an XML file uses a particular schema, then it should conform to the rules specified in the schema.

Visual Studio.NET provides sterling support for XML schemas, and even provides the functionality to automatically generate them from an existing XML document. To generate a schema for our employees.xml file, switch to the data view, right-click in the “tables” list, and choose “Create Schema” from the context menu. A new file, employees.xsd will be added to the project, and this is reflected in the Solution Explorer. Open the file by double-clicking on the node. Figure 9.4 shows the XML schema editor open with the schema for the employees.xml file:



E	employee	(employee)
E	task	(task)
A	name	string
A	title	string
A	salary	string

E	task	(task)
A	priority	string

[Figure 9.4]

XML schema files end with the extension .xsd, as they are written in the “XML Schema Definition Language”, which is in itself a subset of XML. Fortunately Visual Studio.NET provides a graphical editor for creating and editing schemas, so it is not necessary to be able to manually create schemas. The editor is very comprehensive, and allows for full manipulation of schemas according to the W3C specification. The ability to graphically create schemas is obviously a major productivity advantage, especially when working with complex schemas. Even our very simple employees.xml files requires quite a long schema. Clicking on the “XML” toggle button at the bottom left of the editor will reveal the actual XML schema code, which will look similar to listing 9.3.

```
<?xml version="1.0"?>
<xs:schema id="employees"
targetNamespace="http://tempuri.org/employees.xsd"
xmlns:mstns="http://tempuri.org/employees.xsd"
xmlns="http://tempuri.org/employees.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
attributeFormDefault="qualified" elementFormDefault="qualified">
  <xs:element name="employees" msdata:IsDataSet="true"
msdata:Locale="en-ZA" msdata:EnforceConstraints="False">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="employee">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="task" nillable="true" minOccurs="0"
maxOccurs="unbounded">
                <xs:complexType>
```

```
         <xs:simpleContent msdata:ColumnName="task_Text"
msdata:Ordinal="1">
          <xs:extension base="xs:string">
           <xs:attribute name="priority" form="unqualified" type="xs:string"
/>
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
      <xs:attribute name="name" form="unqualified" type="xs:string" />
      <xs:attribute name="title" form="unqualified" type="xs:string" />
      <xs:attribute name="salary" form="unqualified" type="xs:string" />
     </xs:complexType>
    </xs:element>
   </xs:choice>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

The graphical editor certainly takes a lot less effort to understand what the schema actually requires an XML document to conform to. In the case of the employees.xsd schema, the XML document is required to be made up of employee elements. These elements can optionally contain any of the three attributes (marked with an “A” in the editor), “name”, “title” and “salary”, which all contain data of type “string”. Each employee element may also contain a child element (marked with an “E”) of type “task”. The task element is defined in the second block, and also allows the optional inclusion of an attribute, this time called “priority”, of type “string”. The line between the two blocks indicates the parent-child relationship between the two “tables”.

Try out editing the schema by changing the data type of the “salary” attribute in the employee node block, and the “priority” attribute in the task node block to “integer” by choosing the appropriate item from the dropdowns that appear in the fields when they are clicked.

XML & Autocomplete

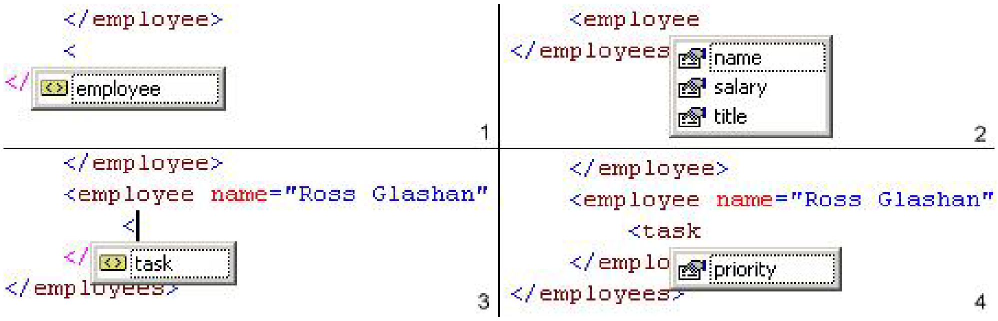
In the HTML editor in Visual Studio, when you type an angled-bracket, indicating the start of a tag, the editor automatically displays a dropdown containing the list of valid tags that can be inserted. The “Autocomplete” technology also takes effect for attributes in HTML, and is a very useful feature that reduces the need to consult HTML reference material. However, since XML is made up of tags that you invent to best represent your data, Autocomplete does not normally take effect in the XML editor. XML schemas are used to define exactly what can and can’t be inserted into an XML document, so when an XML document uses a schema, the IDE will automatically starting using Autocomplete to prompt you with the valid options from the schema when editing the document.

When you had the IDE automatically create the schema for the employees.xml file for you, not only did it create the schema file, but it also “tied” the employees.xml document to the employees.xsd schema. This was done by adding an attribute to the root element, so that it looks as follows:

```
<employees xmlns="http://tempuri.org/employees.xsd">
```

The xmlns attribute is used to specify which schema should be used to validate the document. The value “http://tempuri.org/employees.xsd” is actually an XML namespace (XML namespaces are another part of XML, and were created by the W3C to prevent overlapping XML markups from clashing, but that topic is far beyond the scope of this introduction). You may be wondering where the “tempuri.org” came from, as your site has nothing to do with that domain, but tempuri.org is the default XML namespace, and the one that was automatically chosen and used in the employees.xsd XML schema (you can change this by changing the namespace in both the schema and employees.xml).

With the schema now “attached” to the XML document, Visual Studio is able to now offer intelligent Autocomplete functionality. Figure 9.5 shows the 4 scenarios where Autocomplete displays options – when a tag is opened within the <employees> element (1), when an attribute is expected in the <employee> tag (2), when a tag is opened within an <employee> tag (3), and finally when an attribute is expected in a <task> element (4).



```
    </employee>
    <
</
```

```
    <employee
</employees
```

```
    </employee>
    <employee name="Ross Glashan"
        <
    </
</employees>
```

```
    </employee>
    <employee name="Ross Glashan"
        <task
    </emplo
</employees
```

[Figure 9.5]

Another really useful feature that is included in the XML editor when a schema is being used is that the editor will automatically underline elements or attributes that don’t comply with the schema with a red squiggle, in the same way the invalid VB code is underlined. This automatic validation of the document against the schema is very useful for quickly spotting errors in the XML.

Conclusion

Armed with enough XML knowledge to be dangerous, you should now be able to understand and identify the uses and purpose of the examples give in this chapter of how to manipulate XML data using the .NET BCL. However, as has already been mentioned, XML is a much bigger topic than this chapter can cover, and not only is this introduction to XML very brief, and only intended as a primer, but the support of XML in the .NET framework also extends way beyond what will be covered in this chapters. XML is deeply rooted in the remoting classes, web services framework, and ADO.NET, amongst other things, and this chapter only offers a brief look at basic XML manipulation in .NET.

Using XML in .NET

Reading XML Data

The System.Xml namespace includes numerous classes for dealing with XML data, but the simplest way of reading XML data from an XML file is by using the System.Xml.XmlTextReader class. This class provides functionality for XML files that is not dissimilar to what the System.Data.DataReader class does for relational data – it provides a forward-only, read-only method of reading data from an XML file. It contains takes a no-frills approach, and doesn't automatically valid the XML against the schema or any other fancy extras, but provides the basic functionality for quickly and easily reading XML data.

To demonstrate how to perform simple XML manipulation, this section will be using the XML file outlined in Listing 9.2 that contains a small XML employees database.

Create a new web form and add two controls – a DropDownList and a Button, named "lstEmployees" and "btnView" respectively. In the code-behind, create a new sub-routine, with the code listed in Listing 9.4.

```
Private Sub LoadEmployees()
  LstEmployees.Items.Clear()
  Dim objXML As New XmlTextReader(Server.MapPath("employees.xml"))
  While objXML.Read()
    If objXML.NodeType = XmlNodeType.Element Then
      If objXML.Name = "employee" Then
        lstEmployees.Items.Add(objXML.GetAttribute("name"))
      End If
    End If
  End While
  objXML.Close()
End Sub
```

[Listing 9.4]

This procedure will load the employees.xml file and run through it, adding each employee into the lstEmployees dropdown. The XmlTextReader constructor allows the filename to be passed. Server.MapPath() is used to determine the full physical path of the employees.xml file, which is then passed to the constructor. Alternatively it would be possible to pass no arguments to the constructor and then call the Load() method directly after the declaration like this:

```
objXML.Load(Server.MapPath("employees.xml"))
```

Similar to the DataReader class, the XmlTextReader's Read() method moves to the next node if possible and returns True, although if the end of the file is reached, False is returned. Therefore the While loop iterates through each node until the end of the file is reached.

There is a difference between a "node" and an "element" in XML. Many different items are recognised as nodes. Whitespace between elements (tabs; spaces etc), the XML declaration (<?xml?>), elements (the "tags" eg. <employee>) and comments are all examples of nodes. For this reason it is necessary to distinguish what type of node is currently being dealt with before proceeding. For this purpose, the XmlTextReader exposes the NodeType property, which contains an item from the XmlNodeType enumeration. Figure 9.6 shows the most commonly used items in the enumeration.

Value	**Description**
CDATA	CDATA regions (<![CDATA[]]>)
Comment	Comments (<!-- -->)
Element	Elements (<elementname>)
EndElement	The closing tag of an element (</elementname>)
ProcessingInstruction	XML Processing Instructions (<?instruction?>
Text	Text included as the child of an element (<elementname>Text</elementname>)
Whitespace	Tabs; spaces etc between elements (</elementname>_____<elementname>)
XmlDeclaration	The XML declaration, distinguished from XML processing instructions (<?xml?>)

[Figure 9.6]

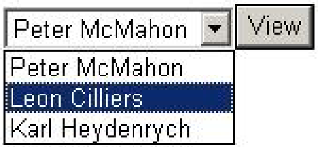
To add the names of the employees to the dropdown list, we only need to access “employee” XML elements. Using a condition to only deal with nodes of type Element ensures that only the elements in the document are processed, but there are elements other than the “employee” element (namely the “employees” and “task” elements), so to further isolate only the “employee” elements, another condition is used – this one ensures that the name of the element is “employee”. Finally, the XmlTextReader exposes a GetAttribute() method that accepts the index of the attribute to be retrieved, or the attribute’s name, or a combination of the attributes name and the namespace, and returns the value from the attribute as a string. In this case, the second overloaded version is used, with the name of the attribute being specified – “name”. The dropdown list’s Items.Add() method is used to add the returned attribute value to the dropdown list.

Once the loop has completed and all the nodes in the document have been covered, it is important that the file is closed. The XmlTextReader’s Close() method oversees this formality.

To test the code, add the following three lines to the Page_Load event handler:

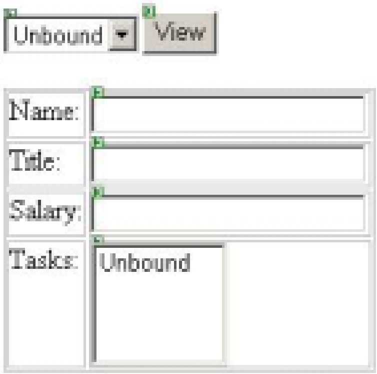
```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  If Not Page.IsPostBack Then
    LoadEmployees()
  End If
End Sub
```

This will ensure that the items are only added to the dropdown the first time the page is loaded – from there onwards, the ASP.NET viewstate functionality will take care of maintaining the state of the dropdown. The following shows how the dropdown should look when the page is loaded in a browser:



Reading Individual Items

Once the user has selected an employee from the dropdown, s/he should be able to view the details of the employee when s/he clicks the “View” button. These details include the name, position and salary of the employee, as well as a list of the employee’s responsibilities. To do this, add three text fields, named txtName, txtTitle and txtSalary, and a ListBox, named lstTasks to the form, as shown in figure 9.6.



[Figure 9.6]

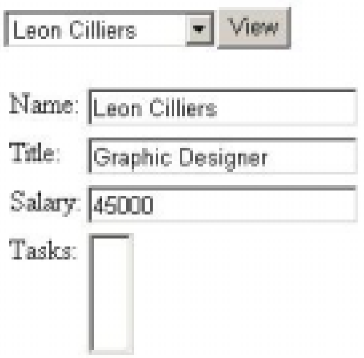
Using the XmlTextReader class to obtain only the values from one specific element is slightly more involved that just reading the entire XML file. To begin with, we’ll only read the values stored in the employee element’s attributes, namely the employee’s name, title and salary. Listing 9.5 shows the code for the Click event handler of btnView:

```
Private Sub btnView_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnView.Click
  Dim objXML As New XmlTextReader(Server.MapPath("employees.xml"))
  While objXML.Read()
    If objXML.NodeType = XmlNodeType.Element Then
      If (objXML.Name = "employee") And (objXML.GetAttribute("name") =
lstEmployees.SelectedItem.Text) Then
        txtName.Text = objXML.GetAttribute("name")
        txtTitle.Text = objXML.GetAttribute("title")
        txtSalary.Text = objXML.GetAttribute("salary")
        Exit While
      End If
    End If
  End While
  objXML.Close()
```

```
End Sub
```

[Listing 9.5]

The XmlTextReader object is created, and the employees.xml file is automatically loaded by passing its path as an argument to the constructor. A loop is again constructed to iterate through each node in the XML file. The first condition checks to see if the current node is an element. If so, a second if statement checks if the name of the element is “employee” (the element name is analogous to the “tag” name) and that the element’s name attribute is the same as the selected item’s text display. If this condition is satisfied, then the TextBox controls are assigned values according to the attributes of the element. Since it is no longer necessary to continue running through the nodes checking for the correct element, since it has already been found, the “Exit While” statement is called to stop the loop. Finally, the XML file is closed. Figure 9.7 shows the output in a browser after one of the employees has been selected:



Name: Leon Cilliers

Title: Graphic Designer

Salary: 45000

Tasks:

[Figure 9.7]

Reading the items into the list of tasks is slightly more tricky. When reading the name, title and salary, all that was required was a simple call to the GetAttribute() method. However, the tasks are not stored as attributes of the employee element – they are child elements, and the XmlTextReader does not expose a specific method for obtaining the text and attribute values from such elements (i.e. there is no method such as GetChildElementValue). Instead, they must themselves be filtered out in the main loop. Since only the tasks for the selected employee are wanted, this entails more than simply checking to see if the node’s Name is “task” – the procedure must keep track of whether or not the loop is currently “inside” the correct employee element, and therefore must add the current task element to the list. Listing 9.6 shows the code to achieve this:

```
Private Sub btnView_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnView.Click
  lstTasks.Items.Clear()
  Dim objXML As New XmlTextReader(Server.MapPath("employees.xml"))
  Dim bInsideSelectedEmployeeNode As Boolean = False
  While objXML.Read()
    If objXML.NodeType = XmlNodeType.Element Then
      If (objXML.Name = "employee") And (objXML.GetAttribute("name") =
lstEmployees.SelectedItem.Text) Then
        txtName.Text = objXML.GetAttribute("name")
        txtTitle.Text = objXML.GetAttribute("title")
        txtSalary.Text = objXML.GetAttribute("salary")
        bInsideSelectedEmployeeNode = True
      ElseIf (objXML.Name = "employee") And
(bInsideSelectedEmployeeNode) Then
        Exit While
      ElseIf (objXML.Name = "task") And (bInsideSelectedEmployeeNode =
True) Then
        Dim strPriority As String = objXML.GetAttribute("priority")
        lstTasks.Items.Add(objXML.ReadString() + " (" + strPriority + ")")
      End If
    End If
  End While
  objXML.Close()
End Sub
```

[Listing 9.6]

The XML file is loaded into the XmlTextReader, and again, a While loop iterates through each node in the file, and checks to see if the current node is an element using an if condition. However, the change comes in that there are now three possible scenarios where code may be executed. The first is if the current node is an “employee” element and its name attribute is equal to the selected name in the DropDownList. Just as before, the three TextBox controls have their Text property assigned to the appropriate attribute value using the GetAttribute() method. However, the addition to this code block is the flagging of the bInsideSelectedEmployeeNode variable to true. The purpose of this is to notify further iterations of the loop that the nodes they are dealing with are children of the correct employee node, and as such should be handled.

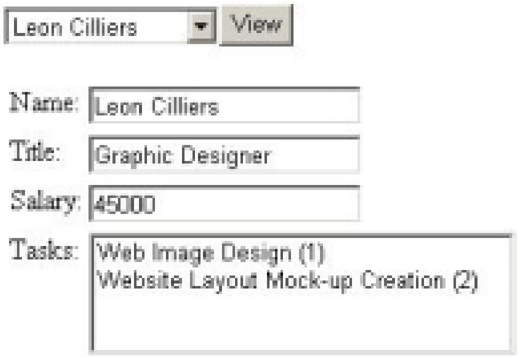
The first ElseIf block checks to see if the current node is an “employee” element, and that the bInsideSelectedEmployeeNode variable is true. If this is the case, then it signals that the loop has already reached the selected employee’s element (which is when bInsideSelectedEmployeeNode is set to true), and that it has finished processing the children of that node, because the next “employee” element has been reached (and employee nodes do not contain other employee nodes), so it is therefore appropriate to exit the loop at this stage, so the “Exit While” statement is executed.

The second ElseIf block deals with the “task” elements that are within the selected “employee” element. The condition checks that the node is a “task” element, and also checks to see if the “task” element is within the selected “employee”, which is signalled by the bInsideSelectedEmployeeNode variable. The ListBox must contain both the task description, and the priority. The priority is stored in an attribute, so it is accessed using the GetAttribute(). However, an element’s text is considered as a separate node, so is not directly accessible, and the XmlTextReader must progress one node in order to access the text. The XmlTextReader class provides a method for exactly this type of situation – ReadString(). This method progresses the object to the next node and returns the node’s Value property as a string.

The ReadString() function probably looks something like this internally:

```
Public Function ReadString() As String
  Me.Read()
  Return Me.Value
End Function
```

Therefore the second line in the block adds the “task” element’s text, along with the priority in brackets. Figure 9.8 shows the final result.



Name: Leon Cilliers
Title: Graphic Designer
Salary: 45000
Tasks: Web Image Design (1)
Website Layout Mock-up Creation (2)

[Figure 9.8]

Manipulating XML Data

Just as for the System.Data classes the DataReader is a “lightweight” class with the sole purpose of reading in data, and there are other classes designed for manipulating data, so too do such variations exist in the System.Xml namespace. The XmlTextReader (and its several other variations that weren’t covered) provides quick and easy access to XML data, but as the last part of the sample application in the previous section demonstrated, once tasks more complicated than just plain reading data are required, the solutions tend to become clumsy and require the use of additional variables to track the read loop’s location in the file. For more “heavy duty” manipulation, the XmlDocument class is provided. This class, as its namesake suggests, is an object-oriented representation of an XML file, and includes a multitude of methods and properties for directly reading from and manipulating XML files.

Reading XML – The XmlDocument vs The XmlTextReader

As a comparison of the XmlDocument’s ability to read XML as compared to the XmlTextReader class, listing 9.7 shows the code to implement the event handler that was shown in listing 9.6 using the XmlTextReader.

```
Private Sub btnView_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnView.Click
  lstTasks.Items.Clear()
  Dim objXMLDoc As New XmlDocument()
  objXMLDoc.Load(Server.MapPath("employees.xml"))
  Dim objXMLNode As XmlNode
  For Each objXMLNode In objXMLDoc.DocumentElement.ChildNodes
    If objXMLNode.Attributes.ItemOf("name").Value =
lstEmployees.SelectedItem.Text Then
```

```
      txtName.Text = objXMLNode.Attributes.ItemOf("name").Value
      txtTitle.Text = objXMLNode.Attributes.ItemOf("title").Value
      txtSalary.Text = objXMLNode.Attributes.ItemOf("salary").Value
      Dim objXMLTaskNode As XmlNode
      For Each objXMLTaskNode In objXMLNode.ChildNodes
        lstTasks.Items.Add(objXMLTaskNode.InnerText + " (" +
objXMLTaskNode.Attributes.ItemOf("priority").Value + ")")
      Next
      Exit For
    End If
  Next
End Sub
```

[Listing 9.7]

The XmlDocument’s constructor does not allow a filename to be passed, so no arguments are passed, and the following line loads the XML from the employees.xml file into the object. A variable of type XmlNode is then declared, and used in a For Each loop that iterates through all the child nodes of the DocumentElement element. The DocumentElement is a reference to the root node, or the element that contains the document, which incidentally in the case of employees.xml is the “employees” element. The ChildNodes property returns an object of type XmlNodeList, which can be looped through using the For Each construct. In each iteration of the loop, the objXMLNode variable is instantiated as a new object of type XmlNode, representing the current node.

The XmlNode object (and consequently objXMLNode) exposes an Attributes property, which is a reference to an XmlAttributeCollection. The ItemOf() property of that class allows an attributed to be selected by name, and it’s value returned using the Value property of the resulting XmlAttribute. The first condition in the loop uses this to check that the “name” attribute of the current element node to ensure that it is the same as the selected name before continuing. If it is, then the “selected” node has been reached, and so the TextBox controls are filled accordingly. However, whereas with the XmlTextReader it was not possibly to simply create another loop to iterate through the selected “employee” element’s child nodes (i.e. its “task” elements), the XmlDocument has that ability, because nodes are exposed as XmlNode objects. Another XmlNode is declared, and another For Each loop is constructed to iterate through the ChildNodes collection of the selected “employee” element. The XmlNode also exposes the InnerText property, which returns the text contained in between the element’s opening and closing tags. The priority is extracted from the appropriate attribute, and both the text and attribute values are added to the list. Finally “Exit For” is called to stop looping through the elements, as the correct one has already been found.

The result is identical to that using the XmlTextReader, and although the code isn’t significantly shorter (2 lines) it is definitely much clearer and more elegant, and shows that for tasks where it is necessary to do more than simply loop through all the nodes in an XML file, the XmlDocument class is a much better option, and this holds true even more when more complex tasks are required.

Modifying XML Data

Where the XmlTextReader’s abilities end at being able to read XML data, for the XmlDocument object, that’s only the start. In addition to being able to load and read XML data, it can also modify, add and delete XML data, and save the changes to file using the same object model that is used for reading the data.

To start off with, we’re going to add the ability to edit the name, title and salary of employees in the employees.xml file. Add a button, btnEditEmployee to the form (preferably in a new row below the “salary” row), and create a new Click event handler for it. Listing 9.8 shows the code that should be used to perform the operation:

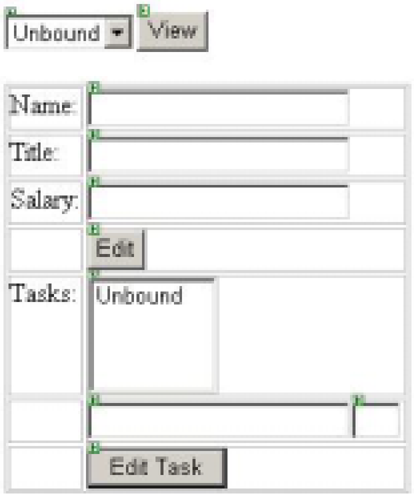
```
Private Sub btnEditEmployee_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnEditEmployee.Click
  Dim objXMLDoc As New XmlDocument()
  objXMLDoc.Load(Server.MapPath("employees.xml"))
  Dim objXMLNode As XmlNode
  For Each objXMLNode In objXMLDoc.DocumentElement.ChildNodes
    If objXMLNode.Attributes.ItemOf("name").Value =
lstEmployees.SelectedItem.Text Then
      objXMLNode.Attributes.ItemOf("name").Value = txtName.Text
      objXMLNode.Attributes.ItemOf("title").Value = txtTitle.Text
      objXMLNode.Attributes.ItemOf("salary").Value = txtSalary.Text
      Exit For
    End If
  Next
  objXMLDoc.Save(Server.MapPath("employees.xml"))
  LoadEmployees()
End Sub
```

[Listing 9.8]

This code is fairly similar to Listing 9.7 (which read items from the XML using the XmlDocument class) for the first part, where it loads the employees.xml file, starts a loop to iterate through all the child nodes of the root node, and checks each iteration to see if the current node is the one that is selected in the ListBox. It is in this code block that the changes occur. Instead of reading from the XmlAttribute objects obtained using the Atributes.ItemOf() property of the objXMLNode object, the code instead assigns the respective values from the TextBox controls into the appropriate attributes. The job tasks are not being edited, so a second loop is not included, and the “Exit For” statement is executed. The next change comes after the loop, where the Save() method of the XmlDocument class is called. This method has several overloaded versions, of which one allows a filename to be passed, which is the one used in the example. Finally the LoadEmployees() method, which was created in the previous section, “Reading XML Data”, is called to refresh the lstEmployees DropDownList, as the user may have changed the name of the selected employee.

Editing the Tasks

Since there is a list of tasks for each employee, editing a specific task posts more of a challenge than simply editing the name, position or salary of an employee. To make editing intuitive, the figure 9.9 shows the form interface with the three additions to enable editing – a TextBox, txtTaskDescription, for holding the task's description; a second TextBox, txtTaskPriority, for holding the task's priority and a Button, btnEditTask, for performing the update.



[Figure 9.9]

The basic procedure for editing is as follows: the user chooses an item from the task ListBox control. When the user clicks an item, the page will post back and display the description and priority of the selected item in the two TextBox controls beneath the ListBox. The user then makes the modifications to the description and/or priority and clicks the "Edit Task" button, which performs the actual update to the XML file.

Firstly, we need to implement the event handler for the lstTasks ListBox's SelectedItemChanged event. To start off with, the AutoPostBack property of the ListBox must be set to True (so that when the user selects a different item, a postback occurs and the two TextBox controls can be filled). Since the ListBox contains both the description and the priority, there is no need to load those two values from the XML file, although there is still some work to be done here, as both value need to be extracted from one string. Regular expressions and the Regex class provide a great solution for this.

> Chapter 7, "User Input Validation" featured a primer on regular expressions which may be helpful if you haven't used them before.

Listing 9.9 shows the event handler for the ListBox, as well as the two supporting functions for extracting the description and priority from the selected item.

```
Private Function ExtractTaskDescription() As String

   Dim objRegEx As New

System.Text.RegularExpressions.Regex("(?<desc>[\w\s-]+) \(?[\d]+\)")
```

```
  Dim objMatch As System.Text.RegularExpressions.Match =
objRegEx.Match(lstTasks.SelectedItem.Text)
  If objMatch.Success Then
    Return objMatch.Groups("desc").Value
  Else
    Return ""
  End If
End Function

Private Function ExtractTaskPriority() As String
  Dim objRegEx As New System.Text.RegularExpressions.Regex("
\((?<priority>[\d]+)\)")
  Dim objMatch As System.Text.RegularExpressions.Match =
objRegEx.Match(lstTasks.SelectedItem.Text)
  If objMatch.Success Then
    Return objMatch.Groups("priority").Value
  Else
    Return "0"
  End If
End Function

Private Sub lstTasks_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles lstTasks.SelectedIndexChanged
  txtTaskDescription.Text = ExtractTaskDescription()
  txtTaskPriority.Text = ExtractTaskPriority()
End Sub
```

[Listing 9.9]

The two functions, ExtractTaskDescription() and ExtractTaskPriority() pull the respective strings from the selected item in the ListBox. Both work in essentially the same way, just using different regular expressions. A new Regex object is created, with the regular expression being passed into the constructor as a string. Next, a Match object is created using the Regex object's Match() method. If the match was successful (i.e. the Regex object managed to successfully parse the string according to the specified regular expression), then the appropriate value from the Match object is returned.

The actual event handler simply calls the two extraction functions and assigns their results to the two respective TextBox controls.

The second part of implementing this editing functionality is to build the event handler for the Button's Click event. This is the code that will actually make the changes in the XML file. Because it is possible that the user changes the description of the task, another procedure is also created to refresh the task list. Listing 9.10 shows both this procedure, and the event handler for btnEditTask.

```
Private Sub LoadTasks()
```

```
  lstTasks.Items.Clear()
  Dim objXMLDoc As New XmlDocument()
  objXMLDoc.Load(Server.MapPath("employees.xml"))
  Dim objXMLNode As XmlNode
  For Each objXMLNode In objXMLDoc.DocumentElement.ChildNodes
    If objXMLNode.Attributes.ItemOf("name").Value =
lstEmployees.SelectedItem.Text Then
      Dim objXMLTaskNode As XmlNode
      For Each objXMLTaskNode In objXMLNode.ChildNodes
        lstTasks.Items.Add(objXMLTaskNode.InnerText + " (" +
objXMLTaskNode.Attributes.ItemOf("priority").Value + ")")
      Next
      Exit For
    End If
  Next
End Sub

Private Sub btnEditTask_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnEditTask.Click
  Dim objXMLDoc As New XmlDocument()
  objXMLDoc.Load(Server.MapPath("employees.xml"))
  Dim objXMLNode As XmlNode
  For Each objXMLNode In objXMLDoc.DocumentElement.ChildNodes
    If objXMLNode.Attributes.ItemOf("name").Value =
lstEmployees.SelectedItem.Text Then
      Dim objXMLTaskNode As XmlNode
      Dim strDescription As String = ExtractTaskDescription()
      For Each objXMLTaskNode In objXMLNode.ChildNodes
        If objXMLTaskNode.InnerText = strDescription Then
          objXMLTaskNode.InnerText = txtTaskDescription.Text
          objXMLTaskNode.Attributes.ItemOf("priority").Value =
txtTaskPriority.Text
          Exit For
        End If
      Next
      Exit For
    End If
  Next
  objXMLDoc.Save(Server.MapPath("employees.xml"))
  LoadTasks()
End Sub
```

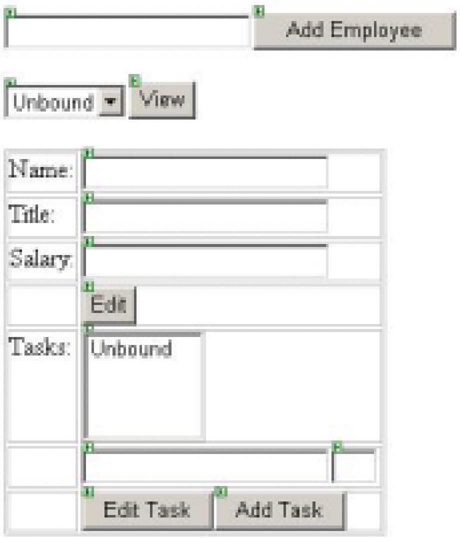
[Listing 9.10]

The btnEditTask Click event handler is identical it the Click handler for btnEditEmployee up until the first If statement in the outer loop. It basically creates a new instance of the XmlDocument class, loads the employees.xml file, starts a loop to iterate through all the “employee” nodes in the XML, and the If condition checks to see if the current “employee” element is the selected on in the lstEmployees ListBox. It is in this code block that the changes occur. Firstly, an XmlNode variable is declared, which will be used in another For Each loop. strDescription is declared as a string and is initialised by calling the ExtractTaskDescription() function. A second loop is started, this time looping through all the “task” nodes beneath the current “employee” element. The If condition checks to see if the current “task” node’s description (stored in between the node’s opening and closing tags as text) is the same as the selected item’s description, and if so, the current node’s InnerText and “priority” attribute are assigned values accordingly. Since the correct node has already been modified, the inner loop is exited using the “Exit For” statement, as is the outer loop, using the same method.

Finally the XML file is saved, using the XmlDocument’s Save() method, and the LoadTasks() procedure is called. This procedure is fairly straightforward. It starts off by clearing the list, and then proceeds to load the employees.xml file and create a loop to iterate through all the “employee” elements. The If condition checks to see if the current node is the selected employee from the ListBox, and if so, proceeds to set up another loop that iterates through all the “task” items, adding each one to the lstTasks ListBox.

Adding XML Data

As with displaying and editing XML data, the XmlDocument class also makes adding XML data extremely painless. However, before we can build the functionality for adding new employees and new tasks to our sample application, we must first add several controls. Firstly, there should be a TextBox (named txtNewEmployeeName) where the user enters the name of the person that is going to be added. Secondly, a button that the user clicks to add the person must be available, and finally a second button must be added allowing the user to add tasks to an employee (the original “edit task” boxes are perfectly suitable for supplying the data, so no new entry fields are needed). Figure 9.10 shows the new form:



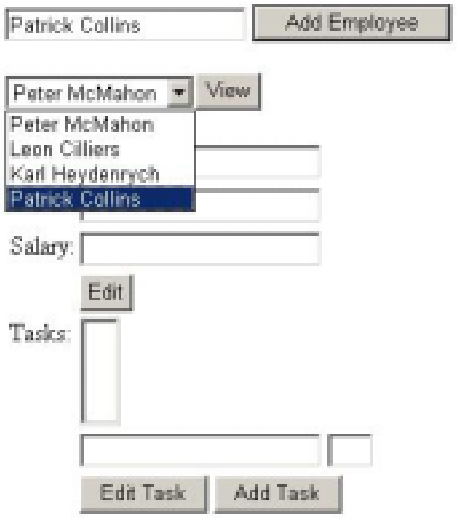
[Figure 9.10]

The code to add a new employee will be placed in the “Add Employee” button’s Click event handler, and will add a new employee element to the XML file, with the name attribute set to the name entered in the txtNewEmployeeName TextBox. Listing 9.11 shows the code for this event handler.

```
Private Sub btnAddEmployee_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnAddEmployee.Click
  Dim objXMLDoc As New XmlDocument()
  objXMLDoc.Load(Server.MapPath("employees.xml"))
  Dim objXMLElement As XmlElement =
objXMLDoc.CreateElement("employee")
  Dim objXMLAttribute As XmlAttribute =
objXMLDoc.CreateAttribute("name")
  objXMLAttribute.Value = txtNewEmployeeName.Text
  objXMLElement.Attributes.Append(objXMLAttribute)
  objXMLAttribute = objXMLDoc.CreateAttribute("title")
  objXMLAttribute.Value = ""
  objXMLElement.Attributes.Append(objXMLAttribute)
  objXMLAttribute = objXMLDoc.CreateAttribute("salary")
  objXMLAttribute.Value = ""
  objXMLElement.Attributes.Append(objXMLAttribute)
  objXMLDoc.DocumentElement.AppendChild(objXMLElement)
  objXMLDoc.Save(Server.MapPath("employees.xml"))
  LoadEmployees()
End Sub
```

[Listing 9.11]

The code starts off by opening the employees.xml file as usual, however, for the most part, this code is entirely different from the editing, or other routines, mainly due to the fact that it is not necessary to find any particular element before a new employee can be added. It is simply added directly onto the “employees” root element. The third line creates a new “employee” element to be stored in a variable of type XmlElement. Take note that this does not actually add the element to the XML file – it simply creates an XML element in memory, ready for manipulation. All employee elements have three attributes: “name”, “title” and “salary”. Since the employee is being added using a separate TextBox for entering the name, and there aren’t fields for adding the other two, it is only possible to assign a value for “name”. However, all three attributes must still be created, because the editing facility expects them to exist. Creating attributes is done in much the same way as elements – declare a variable of type XmlAttribute, and initialise it by calling the CreateAttribute() method of the XmlDocument, and it will return the appropriate object. In this example, only the Value property is used, which sets the value that the attribute will hold. Following this, the Attributes.Append() method of the XmlElement variable is called, passing the XmlAttribute variable as a parameter. This is repeated three times, for the three attributes. Once all the attributes have been added to the in-memory element, the element can then be added to the actual document. We want to add it to the “employees” node, which is the document element (or root node), so the DocumentElement.AppendChild() method of the XmlDocument object is called, where the XmlElement is passed as a variable. All that is left to do is save the changes to file, and to update the employees DropDown by calling the LoadEmployees() routine. Figure 9.11 shows the form loaded in a browser after another entry has been added:



[Figure 9.11]

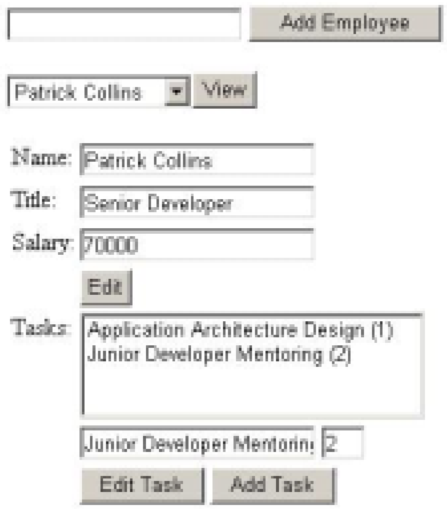
The event handler for the “Add Task” button is slightly more complicated – it first needs to locate the “employee” element that it must add the task under, and then it can add the “task” element. Listing 9.12 shows the code:

```
Private Sub btnAddTask_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAddTask.Click
   Dim objXMLDoc As New XmlDocument()
   objXMLDoc.Load(Server.MapPath("employees.xml"))
   Dim objXMLNode As XmlNode
```

```
    For Each objXMLNode In objXMLDoc.DocumentElement.ChildNodes
      If objXMLNode.Attributes.ItemOf("name").Value =
  lstEmployees.SelectedItem.Text Then
        Dim objXMLTaskElement As XmlElement =
  objXMLDoc.CreateElement("task")
        Dim objXMLAttribute As XmlAttribute =
  objXMLDoc.CreateAttribute("priority")
        objXMLAttribute.Value = txtTaskPriority.Text
        objXMLTaskElement.Attributes.Append(objXMLAttribute)
        objXMLTaskElement.InnerText = txtTaskDescription.Text
        objXMLNode.AppendChild(objXMLTaskElement)
        Exit For
      End If
    Next
    objXMLDoc.Save(Server.MapPath("employees.xml"))
    LoadTasks()
  End Sub
```

[Listing 9.12]

The code again starts off by loading the XML file, and continues the same way as the “Edit Task” code by creating a For loop to iterate through the element, and having an If condition to see if the current node is the selected “employee” element. Inside the If block is where the changes occur. Firstly, a new “task” XmlElement is created using the CreateElement() method from the XmlDocument. Next, a new XmlAttribute is created to represent the “priority” attribute of the “task” element. Its Value property is set to the Text in the txtTaskPriority TextBox. The attribute is then added to the element by using the Attributes.Append() method, again passing the XmlAttribute object as a parameter. The “task” element stores the task priority in an attribute, but the description is included between the “task” tags. To insert this text, the InnerText property is assigned the value from the txtTaskDescription TextBox. Finally, the new XmlElement (objXMLTaskElement) is added to objXMLNode, which is an XmlElement representing the current node in the loop (i.e. the selected employee node). The loop is exited, the file is saved and the tasks are reloaded. Figure 9.12 shows the page is a browser after a couple of tasks have been added to a new employee (Patrick Collins):



Name: Patrick Collins
Title: Senior Developer
Salary: 70000

Tasks:
Application Architecture Design (1)
Junior Developer Mentoring (2)

Junior Developer Mentorin 2

[Figure 9.12]

Deleting XML Data

With adding, editing and displaying XML data using the XmlDocument class presented, there's only one major section left – deleting data. Fortunately deleting nodes is as simple as a single method call. To demonstrate deleting nodes in our sample application, add two buttons – one next to the "View" button, named btnDeleteEmployee and another next to the "Add Task" button, named btnDeleteTask. Listing 9.13 shows the event handler for the first button, btnDeleteEmployee, which deletes the selected employee.

```
Private Sub btnDeleteEmployee_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnDeleteEmployee.Click
  Dim objXMLDoc As New XmlDocument()
  objXMLDoc.Load(Server.MapPath("employees.xml"))
  Dim objXMLNode As XmlNode
  For Each objXMLNode In objXMLDoc.DocumentElement.ChildNodes
    If objXMLNode.Attributes.ItemOf("name").Value =
lstEmployees.SelectedItem.Text Then
      objXMLDoc.DocumentElement.RemoveChild(objXMLNode)
      Exit For
    End If
  Next
  objXMLDoc.Save(Server.MapPath("employees.xml"))
  LoadEmployees()
End Sub
```

[Listing 9.13]

As per normal, the code starts off by opening the XML and starts a loop iterating through all the "employee" elements, searching for the selected one using the If statement. When the If statement evaluates True and the loop has reached the selected employee element, then the following line is executed:

```
objXMLDoc.DocumentElement.RemoveChild(objXMLNode)
```

All the “employee” elements are stored as direct children of the “employees” document element (root element). The objXMLNode object is the XmlNode object representing the current node, which is the selected “employee”. The XmlDocument’s DocumentElement.RemoveChild() method is called, passing the objXMLNode XmlNode object as a parameter, resulting in that node being removed from the “employees” element, along with all the “employee” node’s children (“task” nodes). The loops are exited, the file saved and the employees list reloaded.

Listing 9.14 shows the code for the btnDeleteTask event handler:

```
Private Sub btnDeleteTask_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnDeleteTask.Click
  Dim objXMLDoc As New XmlDocument()
  objXMLDoc.Load(Server.MapPath("employees.xml"))
  Dim objXMLNode As XmlNode
  For Each objXMLNode In objXMLDoc.DocumentElement.ChildNodes
    If objXMLNode.Attributes.ItemOf("name").Value =
lstEmployees.SelectedItem.Text Then
      Dim objXMLTaskNode As XmlNode
      Dim strDescription = ExtractTaskDescription()
        For Each objXMLTaskNode In objXMLNode.ChildNodes
          If objXMLTaskNode.InnerText = strDescription Then
            objXMLNode.RemoveChild(objXMLTaskNode)
            Exit For
          End If
        Next
      Exit For
    End If
  Next
  objXMLDoc.Save(Server.MapPath("employees.xml"))
  LoadTasks()
End Sub
```

[Listing 9.14]

This event handler again follows the regular approach for dealing with “task” nodes – it opens the XML file, then sets up a loop iterating through all the “employee” elements. A second loop is set up to iterate through all the “task” elements of the selected employee (which is found using the first If condition). In this loop, the program is searching for the selected task’s element. The If statement compares the current “task” node’s text (which contains the description) to the description in the tasks ListBox. Inside this If code block, the current “employee” element (represented by the objXMLNode object) call’s its RemoveChild() method, and passes the current “task” element, objXMLTaskNode. The loops are then exited, the file saved, and the tasks list reloaded.

In all these examples, the code for loading the XML document and finding specific elements has been repeated to avoid unnecessary complexity when learning the basics of the XmlDocument and XmlTextReader classes. However, in a real application it would be advisable to reuse code, such as the loading and saving of documents, as well as searching for elements by building custom functions that return XmlElement objects, ready for manipulation.

Using the XmlDocument In-Memory Only

Since XML isn't intended as a replacement for RDBM systems, storing actual data to disc in XML is generally not done for anything other than configuration files. For the most part, XML data will be manipulated in-memory, such as from the result of a raw web service call, or after XML has been obtained from a DataSet from a database. The XmlDocument class includes several methods for loading in-memory XML data. The Load() method includes 4 overloaded versions, of which one accepts a filename as a string, which was used in all the previous examples. However, it can also accept a Stream object, which could be a MemoryStream. An entirely different method, LoadXml() is probably the more common way of accessing in-memory XML data though – it allows actual XML data to be passed to it as a string. For example, the following code will load a piece of XML into a new XmlDocument object and display the value of the first attribute of the first child element of the root node in a Label, lblAttribute1:

```
Dim objXMLDoc As New XmlDocument()
objXMLDoc.LoadXml("<employees><employee name=""Peter McMahon""
title=""Developer"" salary=""60000""></employee></employees>")
lblAttribute1.Text =
objXMLDoc.DocumentElement.ChildNodes.Item(0).Attributes.Item(0).Val
ue
```

Whether XML is loaded from a file using the Load() method, or LoadXml() is used, the functionality of the XmlDocument object remains the same.

Obviously manipulating the XmlDocument is useless if it's not possible to obtain the final XML when you're finished. For this purpose, the InnerXml property of the XmlDocument class is available, which returns the XML as a string. This can then be assigned to a string variable, like this:

```
Dim strXML As String = objXMLDoc.InnerXml
```

XSL Transformations

XSL (or eXtensible Stylesheet Language) often plays a crucial role in the display of XML data. Chapter 12, "Using the Advanced User Controls" details the usage of the Xml Web Control, which makes doing XSL transformations extremely simple. That chapter also includes a brief primer on XSL, as well as information on using the XslTransform class to perform transforms, along with the Xml Web Control.

Conclusion

XML is an extremely large, and extremely important topic. The .NET Framework provides extensive support for XML, and a solid working knowledge of how to use

XML in ASP.NET is important. This chapter has provided a primer on XML to get you to speed on the fundamentals, and guided you through the basics of manipulating XML data using the XmlDocument class. This should certainly not be the end of your learning – many XML and XML-related topics haven't even been mentioned or have been given very brisk overviews, such as XPath, XSL, XML schemas and namespaces. ADO.NET provides very deep intrinsic support for XML, as do the serialization classes, used for converting .NET to XML, and then back again. Most of the .NET configuration files use XML (for example, Web.config), so using the .NET XML parsing capabilities, it is possible to build tools to manipulate the configuration files from an ASP.NET applications, which is potentially very powerful. With XML, the opportunities for expansion and usage are very diverse, and hopefully this chapter has prepared you to experiment with how XML can help you build better, more elegant solutions to problems.

Chapter 10

File IO

In today's world of development, there are many ways to store data and information, ranging from advanced relational data systems such as SQL Server, to the hierarchical data text-based XML format. However, sometimes when designing an application it makes more sense to use one of the oldest methods of storing computer data – the file system. The Windows filesystem provides a very easy-to-use method of storing data in a hierarchical structure. Using the filesystem also provides several inherent advantages from the point of view of resource access control and security because of the Windows security model and NTFS.

The .NET Framework provides a comprehensive set of classes in the System.IO namespace for manipulating files and folders in the filesystem. Files and folders can easily be created, deleted, copied, moved and created. The relevant classes also encapsulate intrinsic NTFS permissions for both files and folders, along with file and folder attributes, and other useful information.

Basic File and Folder Operations

The four primary classes for basic manipulation of files and folders are the System.IO.File, System.IO.FileInfo, System.IO.Directory and System.IO.DirectoryInfo classes respectively. These classes encapsulate most functions that could conceivably be performed on files and folders and are very comprehensive. All the available methods in the File and Directory classes are Shared, or static, and thus instantiating a File or Directory object is not required when working with theses classes.

The File and Directory classes share many of the same method names, which perform essentially the same operations. These include methods for checking the existence of a file or folder, moving a file or folder, or retrieving the "last accessed" and "last modified" dates. However, despite some similarities in functionality, the File and Directory classes do each provide numerous specialized methods that apply onto to one of the two. For example, only the Directory class has the ability to create other directories, or list files in a directory.

Copying and Moving Files and Folders

Both the File and Directory classes provide a "Move" method that allows files and directories to be moved in the filesystem. The Move() method takes two string parameters in both cases – the first specifying the location of the file or folder to be moved, and the second specifying that the location that the file or folder must be moved to, including the filename. For example, the following line of code will move the file "c:\test.txt" to "c:\windows\test.txt".

```
System.IO.File.Move("c:\test.txt", "c:\windows\test.txt")
```

Obviously such code could appear in the Page_Load event, or any event caused by a user's action, such as the clicking of a button. This code could even be executed in the Global.asax file in an event handler, such as the session's Start event. When combined with other File class methods, the Move method could be used for creating a powerful web storage administration front end. However, the important aspect that needs to be acknowledged is that the second parameter specifies a file path, not just a directory, as might be expected. If a directory is specified instead of a full file path, an exception will occur. Another important point to consider is that the Move method does not automatically create the specified file path except for the actual file, and if a file path that does not exist is entered, an exception will also be raised. For example, if the second parameter is "c:\testdir\test.txt", and c:\testdir does not exist, the method will fail.

The Directory class's Move method operates similarly to the File.Move() method. It too accepts 2 parameters, although in this case the parameters specify the folder to be moved, and where the folder must be moved to respectively. As with the File.Move() method, the 2nd parameter must specify the full path that should be created, not just the path to the folder that must be the parent of the moved folder. The following code will move the folder c:\testdir to c:\windows\testdir.

```
System.IO.Directory("c:\testdir", "c:\windows\testdir")
```

Again, as with the File.Move() method, the path specified in the 2nd parameter must already exist, and the method will not automatically create the path specified.

When a directory is copied, all its child files and folders are maintained in their current structural form and copied along with the folder, maintaining their positions in the folder's structure.

The File.Copy() method operates almost identically to the File.Move() method, except that it does not move the file in question, but rather creates a copy of it. It too requires a full existing file path to be entered as its second parameter. The following would copy the file c:\test.txt to the c:\windows directory:

```
System.IO.File.Copy("c:\test.txt", "c:\windows\test.txt")
```

Checking if a File or Folder Exists

Both the File and Directory classes expose an Exists() method, which returns a Boolean value. The sole parameter that is passed is a string value of the path to check. The return value is true if the path exists, and false if it doesn't. The following code will check to see if the directory c:\testdir exits:

```
If System.IO.Directory.Exists("c:\testdir") = True Then
 'Code to execute
End If
```

The code could also easily be modified to check if the file c:\test.txt exists:

```
If System.IO.File.Exists("c:\test.txt") = True Then
 'Code to execute
End If
```

The Exists() method is particularly useful in situations when an exception might be thrown if an operation is performed involving a file or folder that is assumed to exist, but doesn't. For example, the following code will first check that the folder c:\testdir and the file c:\test.txt exists before moving the file into the directory:

```
If (System.IO.Directory.Exists("c:\testdir") = True) And
(System.IO.File.Exists("c:\test.txt") = True) Then
 System.IO.File.Move("c:\test.txt", "c:\testdir\test.txt")
End If
```

However, this code could be advanced by automatically creating the directory if it doesn’t already exist.

Creating Folders

The Directory class exposes the CreateDirectory() method for creating new folders. The method accepts one string parameter specifying the path of the directory to be created. An instance of the DirectoryInfo() class is returned, which includes details about the newly created directory.

The CreateDirectory() method allows multiple levels of directories to be created using one call i.e. the immediate parent of the created directory does not need to exist – the method will create the entire path automatically. For example, if you call the method to create c:\testdir\anotherdir, if c:\testdir doesn’t already exist it will be created automatically. If the method is called to create a directory that already exists, the DirectoryInfo class that is returned simply returns the values for the specified directory, and no exception is raised.

To simply create the directory c:\testdir, the following call would be used:

```
System.IO.Directory.CreateDirectory("c:\testdir")
```

This could be used in conjunction with the Move() or Copy() methods to ensure that a path exists:

```
If System.IO.Directory.Exists("c:\testdir") = False Then
 System.IO.Directory.CreateDirectory("c:\testdir")
End If
System.IO.File.Move("c:\test.txt", "c:\testdir\test.txt")
```

This code ensures that the c:\testdir directory exists before moving the file c:\test.txt to it. However, because the CreateDirectory() method does not raise an exception if the specified folder already exists, it would also be possible to rewrite the above code example as follows:

```
System.IO.Directory.CreateDirectory("c:\testdir")
System.IO.File.Move("c:\test.txt", "c:\testdir\test.txt")
```

The DirectoryInfo class that is returned provides properties and methods for handling the newly created directory. The section “Folder and File Information” deals specifically with manipulating the DirectoryInfo class, but the following code will store an instance of the DirectoryInfo class for the newly-created folder c:\testdir:

```
Dim objDirInfo As System.IO.DirectoryInfo =
System.IO.Directory.CreateDirectory("c:\testdir")
```

Deleting Files and Folders

Both the File and Directory classes implement a Delete() method. In the case of the File class, the Delete() method requires one string parameter to be passed specifying the path of the file to be deleted. The function doesn’t return a value.

However, the Directory class overloads 2 versions of the Delete() method – the first takes only a string parameter, whilst the second takes both a string parameter, and a boolean, which specifies whether to recursively delete the contents of the folder.

To delete the file c:\test.txt, the following statement could be used:

```
System.IO.File.Delete("c:\test.txt")
```

Likewise the directory c:\testdir could be deleted by executing the following statement:

```
System.IO.Directory.Delete("c:\testdir")
```

However, if the specified folder (c:\testdir, in this case) is not empty, an IOException error will be raised. The folder must be completely empty (both of files and subfolders) before it can be deleted using this method. To avoid the error being thrown by recursively emptying the folder before deletion, the 2nd overloaded version of the method can be used. This version allows the first parameter to specify the path as a string, and the second parameter is a boolean specifying whether or not the folder should be recursively emptied. If this value is false, and the folder contains any items, the IOException error will still be raised, but if the value is true, then the folder will be emptied completely then deleted without any errors.

To delete the c:\testdir folder and any contents contained within, the following statement could be used:

```
System.IO.Directory.Delete("c:\testdir", True)
```

File and Folder Information

The File, FileInfo, Directory and DirectoryInfo all provide properties and methods for obtaining information about files and folders. Such information includes creation, “last modified” and “last accessed” dates, attributes, lists of child objects and others. The classes also provide appropriate methods for manipulating this information.

File vs FileInfo; Directory vs DirectoryInfo

The primary difference between the –Info classes and their non –Info equivalents is that the –Info classes are instantiated and represent a particular file or folder on the filesystem, whereas the non –Info classes are not instantiated, and provide Shared methods for accessing individual aspects of specific files or folders. In other words, the two non –Info classes do not store state, and the file or folder referenced must be explicitly passed for every method call, but the –Info classes store the path of the file or folder that they are dealing with, so method calls and properties do not require the path to be passed each time.

The methods and properties exposed by these four classes are very similar, although there are several subtle differences in certain instances, but cumulatively provide the following functionality for obtaining and manipulating related information about files and folders:

- Access/set file attributes
- Access/set directory attributes
- Access/set file creation time
- Access/set directory creation time

- Access/set file “last accessed” & “last modified” time
- Access/set directory “last accessed” & “last modified” time
- Access file size
- List subdirectories in a directory
- List files contained in a directory

Creating FileInfo and DirectoryInfo Classes

As has been mentioned, both these classes need to be instantiated, and each instance refers to a specific file or folder. In both cases, the constructor requires that a string parameter is passed, containing the path of the file or folder. The following code will create a variable “objFileInfo” of type FileInfo that relates to the file c:\test.txt:

```
Dim objFileInfo As New System.IO.FileInfo("c:\test.txt")
```

Likewise the following code could be used to create a new DirectoryInfo object referring to the folder c:\testdir:

```
Dim objDirInfo As New System.IO.DirectoryInfo("c:\testdir")
```

Creation, Access and Modification Times

All 4 file and directory access classes provide methods and properties for both accessing and setting the various time values that are associated with files and folders. The –Info classes expose them as properties, whereas the non –Info classes provide methods to access these time values.

Creation Time

To access the creation time of a file or folder using the File and Directory classes, the GetCreationTime() method should be used. This method takes one string parameter with the path of the file or folder, and returns a DateTime value.

The following code would output the creation time of the file c:\test.txt to a label:

```
Label1.Text = System.IO.File.GetCreationTime("c:\test.txt").ToString("g")
```

GetCreationTime() returns a DateTime object, and the ToString() method is used to convert the DateTime to a string for display. The creation time of a directory could be assigned similarly:

```
Label1.Text =
System.IO.Directory.GetCreationTime("c:\testdir").ToString("g")
```

To set the creation time of a file or directory, the respective SetCreationTime() method must be used. This method accepts a string parameter representing the path of the file or directory and a second parameter of type DateTime specifies the date and time of the new creation time. The following code will assign the creation time of the file c:\test.txt to the current date and time:

```
System.IO.File.SetCreationTime("c:\test.txt", DateTime.Now)
```

This could easily be modified to apply to a directory:

```
System.IO.Directory.SetCreationTime("c:\testdir", DateTime.Now)
```

To access and set the creation time of a file or folder using the FileInfo and DirectoryInfo classes the CreationTime property should be used. This property is of type DateTime, and could be used as follows to obtain the creation time of a file and then modify it:

```
Dim objFileInfo As New System.IO.FileInfo("c:\test.txt")
Label1.Text = objFileInfo.CreationTime.ToString("g")
objFileInfo.CreationTime = DateTime.Now
```

The DirectoryInfo class can be manipulated just as easily:

```
Dim objDirInfo As New System.IO.DirectoryInfo("c:\testdir")
Label1.Text = objDirInfo.CreationTime.ToString("g")
objDirInfo.CreationTime = DateTime.Now
```

Last Accessed and Last Modified Times

As with the creation time, the last accessed and last modified values associated with files and folders can be accessed and set using both the –Info and non –Info classes.

The File and Directory classes provide the GetLastAccessTime() and SetLastAccessTime() methods for manipulating the “last accessed” time of files and folders respectively, and the GetLastWriteTime() and SetLastWriteTime() methods handle the “last modified” time. As with the creation time methods, the Get- methods accept a string parameter specifying the path of the file or folder, and return a DateTime value, while the Set- methods accept a string parameter with the path of the file or folder, and a second parameter of type DateTime.

The following code demonstrates the use of the GetLastAccessTime() and SetLastWriteTime() methods for the File and Directory classes:

```
Label1.Text = System.IO.File.GetLastAcceessTime("c:\test.txt")
System.IO.Directory.SetLastWriteTime("c:\testdir", DateTime.Now)
```

This code places the last accessed time of the file c:\test.txt into a label and sets the last modified time of the c:\testdir folder to the current time.

Again, as with the creation time functionality, the FileInfo and DirectoryInfo classes provide properties for accessing and modifying the last accessed and last modified values of files and folders. The LastAccessTime and LastWriteTime properties are of type DateTime, and are used in the same way as with the CreationTime property:

```
Dim objDirInfo As New System.IO.DirectoryInfo("c:\testdir")
Label1.Text = objDirInfo.LastAccessTime.ToString("g")
objDirInfo.LastWriteTime = DateTime.Now
```

Directory File and Subdirectory Lists

The Directory and DirectoryInfo classes each provide 3 methods for accessing files and subdirectories in a specified directory. The GetDirectories() and GetFiles() methods of the Directory class accept a string parameter passing the parent folder path and both return an array of strings holding either the subdirectories or files contained in the specified folder. The DirectoryInfo class’s GetDirectories() and GetFiles()

methods do not take any parameters, but return an array of DirectoryInfo or FileInfo objects. The third relevant method of the Directory class is GetFileSystemEntries(), which accepts a string parameter containing the path of the folder and returns an array of strings holding the names of all the subdirectories *and* files of the specified folder. The DirectoryInfo class's GetFileSystemInfos() method doesn't take an parameters, but returns an array of FileSystemInfo classes. The FileSystemInfo class is a generic – Info class for dealing with either directories or files, and incidentally both the DirectoryInfo and FileInfo classes derive from the FileSystemInfo class.

The following code demonstrates the use of the GetFiles() method of the Directory class by filling a listbox control, lstFiles, with the names of the files in the directory c:\testdir:

```
Dim strFiles As String() = System.IO.Directory.GetFiles("c:\testdir")
Dim i As Integer
For i = 0 To strFiles.Length – 1
 lstFiles.Items.Add(strFiles(i))
Next
```

Since a regular array is returned, a For Each loop could also be used to iterate through the files:

```
Dim strFiles As String() = System.IO.Directory.GetFiles("c:\testdir")
Dim strFile As String
For Each strFile In strFiles
 lstFiles.Items.Add(strFile)
Next
```

The GetDirectories() method could be used fairly similarly to fill the list lstFolders with the list of subdirectories in c:\testdir:

```
Dim strDirs As String() = System.IO.Directory.GetDirectories("c:\testdir")
Dim strDir As String
For Each strDir In strDirs
 lstDirs.Items.Add(strDir)
Next
```

The GetFileSystemEntries() method which returns both the subdirectories and files contained within the folder can also be easily used to populate a listbox:

```
Dim strEntries As String() =
System.IO.Directory.GetFileSystemEntries("c:\testdir")
Dim strEntry As String
For Each strEntry In strEntries
 lstEntries.Items.Add(strEntry)
Next
```

All three of these methods provide a second overloaded version that allows 2 string parameters to be passed – the first still the path of the folder, but the second specifies criteria for limiting the items returned in the array based on file or folder name. The criteria, or filter, that is passed uses the same format as the command line in Windows – '*.*' means all files; '*.aspx*' means all ASP.NET UI and codebehind files; 'm*' means all files beginning with 'm', and so forth.

The following code populates a listbox with ASP.NET UI files in the c:\testdir directory:

```
Dim strFiles As String() = System.IO.Directory.GetFiles("c:\testdir",
"*.aspx")
Dim strFile As String
For Each strFile In strFiles
 lstFiles.Items.Add(strFile)
Next
```

As has been mentioned, this second parameter can be used with the GetDirectories() and GetFileSystemEntries() methods as well.

The DirectoryInfo class's implementation of the GetFiles(), GetDirectories() and GetFileSystemInfos() methods differ from those of the equivalent methods in the Directory class, but can also easily be used to populate a listbox. The following code fills a listbox with the names of the files within the c:\testdir folder using the DirectoryInfo class:

```
Dim objDirInfo As New System.IO.DirectoryInfo("c:\testdir")
Dim objFile As System.IO.FileInfo
For Each objFile In objDirInfo.GetFiles()
 lstFolders.Items.Add(objFile.FullName)
Next
```

This implementation lists the files including their full paths. The FileInfo class provides the Name property, as opposed to the FullName property, which returns only the file's name and not its path (so Name will return 'test.txt' rather than 'c:\testdir\test.txt'). The GetFiles() method has a second overloaded version which accepts a string parameter containing a filter for the listed files, so it is possible to call the method and have it return only .aspx files etc. The following code fills an array of FileInfo objects with all the ASP.NET UI files in a directory:

```
Dim objFileInfo As System.IO.FileInfo() = objDirInfo.GetFiles("*.aspx")
```

The GetFileSystemInfos() method is similar to the Directory class's GetFileSystemEntries() method in that it returns a list of both subdirectories and files contained in a specific directory. The return is an array of FileSystemInfo objects. The FileSystemInfo class is the parent class of the DirectoryInfo and FileInfo classes, and provides functionality applicable to both folders and files. This includes the times associated with files (CreationTime; LastAccessTime; LastWriteTime), the ability to access Attributes, the Delete() method, and others. To populate a list of both the subdirectories and files in a specific directory using the GetFileSystemInfos() method, the following code can be used:

```
Dim objDirInfo As New System.IO.DirectoryInfo("c:\testdir")
Dim objFileSystemInfo As System.IO.FileSystemInfo
For Each objFileSystemInfo In objDirInfo.GetFileSystemInfos()
 lstFilesAndFolders.Items.Add(objFileSystemInfo.FullName)
Next
```

Again the GetFileSystemInfos() method has a second overloaded version allowing a filter for the returned items to be specified.

File Size

The FileInfo class provides the Length property that contains the size of the file that the FileInfo instance is associated with in bytes. Unfortunately the File class doesn’t provide a method, so an instance of the FileInfo class is required to obtain the size of a file, and neither the Directory nor DirectoryInfo classes provide a method or property for accessing the size of a directory and its contents.

The following code sets the size of the file c:\test.txt to labels, the first in bytes, the second in kilobytes and the third in megabytes:

```
Dim objFileInfo As New System.IO.FileInfo("c:\test.txt")
lblBytes.Text = objFileInfo.Length.ToString()
lblKBytes.Text = (objFileInfo.Length \ 1024).ToString()
lblMBytes.Text = (objFileInfo.Length \ 1024 \ 1024).ToString()
```

To alleviate the problem of the Directory class not exposing a method for obtaining the size of a directory, it is possible to create your own function that does so using the FileInfo class. The following subroutine is a recursive function that allows the folder path to be specified, along with a variable to store the total size of the folder in:

```
Private Sub GetFolderSize(ByVal folder As String, ByRef size As Integer)
 Dim strDirs As String() = System.IO.Directory.GetDirectories(folder)
 Dim i As Integer
 For i = 0 To strDirs.Length - 1
  GetFolderSize(strDirs(i), size)
 Next
 Dim strFiles As String() = System.IO.Directory.GetFiles(folder)
 For i = 0 To strFiles.Length - 1
  size += New System.IO.FileInfo(strFiles(i)).Length
 Next
End Sub
```

This subroutine would be called as follows to obtain the total size of the c:\testdir directory:

```
Dim size As Integer
GetFolderSize("c:\testdir", size)
```

The variable ‘size’ would contain the total size of the c:\testdir folder in bytes after the code is executed. The GetFolderSize() sub calls itself for every folder within the specified “root” folder, allowing the sub to accurately return the size of any folder’s contents, no matter how many directory levels it has. For every file in any particular folder, the ‘size’ variable is incremented by the size of the file.

Attributes

Every file or folder in the Windows filesystem has attributes associated with it. These attributes may store information such as whether a file or folder is read only, hidden, compressed or encrypted, amongst others. The File, FileInfo and DirectoryInfo classes allow you to access and manipulate this information. The File class provides the GetAttributes() method which returns a value from the

System.IO.FileAttributes enumeration. This is a bit flag enumeration, meaning that each value in the enumeration is specifically assigned an Integer value so that more than one value can be selected (because a file can be marked as both hidden and archive, for example). The File class's SetAttributes() method allows for attributes to be set for a particular file. The FileInfo and DirectoryInfo classes provide an "Attributes" property, which allows read and write access to the associated FileAttributes enumeration.

Since the FileAttributes enumeration is bit flagged, it is slightly more involved to deal with when requesting specific items, but to output a basic list of the attributes set for a specific file to a label is fairly simple:

```
Label1.Text = System.IO.File.GetAttributes("c:\test.txt").ToString()
```

This will return a comma-separated string list of the attributes applicable to the file c:\test.txt. The possible options from the FileAttributes enumeration are listed in Table 10.1.

Member:	Description:
Archive	Is the file marked for archival purposes?
Compressed	Is the file compressed?
Device	
Directory	Is the file a directory?
Encrypted	Is the file encrypted?
Hidden	Is the file hidden?
Normal	Set if no other attributes apply
NotContentIndexed	Is the file marked as not available for indexing?
Offline	Is the file offline?
Readonly	Is the file readonly?
ReparsePoint	Does the file contain a reparse point?
SparseFile	Is the file sparse (typically large; mainly zeros)
System	Is the file marked as a system file?
Temporary	Is the file a temporary file?

[Table 10.1 – System.IO.FileAttributes enumeration members]

To receive a boolean of whether one of the enumeration members applies to a specific file (such as 'Is a file hidden?'), the following code can be used:

```
Dim objFileAttributes As System.IO.FileAttributes =
System.IO.File.GetAttributes("c:\test.txt")
Dim bHidden As Boolean = ((objFileAttributes And
System.IO.FileAttributes.Hidden) <> 0)
```

The boolean bHidden will contain a value of True if the file c:\test.txt is hidden, and False is it isn't. This method can obviously be applied to any value in the FileAttributes enumeration.

To set the attributes of a file using the File's SetAttributes() method a string containing the path of the file and values from the FileAttributes enumeration must be passed. The following will mark the file c:\test.txt as read-only:

```
System.IO.File.SetAttributes("c:\test.txt",
System.IO.FileAttributes.ReadOnly)
```

Calling SetAttributes() will automatically remove all attributes from the file and apply only the attributes specified. In the above case, the file c:\test.txt will only be flagged as read-only, regardless of what other flags it had previously. To specify numerous attributes, the bitwise Or operator should separate the specific values. Below shows code to set the attributes of the c:\test.txt to both hidden and read-only:

```
System.IO.File.SetAttributes("c:\test.txt",
System.IO.FileAttributes.ReadOnly Or System.IO.FileAttributes.Hidden)
```

To avoid all the previous attributes of a file being reset it is possible to specify that you want to apply the current attributes *and* set another one:

```
System.IO.File.SetAttributes("c:\test.txt",
System.IO.File.GetAttributes("c:\test.txt") Or
System.IO.FileAttributes.ReadOnly)
```

This will ensure that the file c:\test.txt is read-only, whilst maintaining its current attributes. The Xor operator can be used to toggle whether a specific attribute is applied or not. For example, the following code will set the file c:\test.txt to read-only if it isn’t read-only, but will set it to be non-read-only if it is read-only:

```
System.IO.File.SetAttributes("c:\test.txt",
System.IO.File.GetAttributes("c:\test.txt") Xor
System.IO.FileAttributes.ReadOnly)
```

The FileInfo and DirectoryInfo classes provide the Attributes property for accessing and setting attributes. All the methods of accessing and setting attributes for the GetAttributes() and SetAttributes() methods of the File class still apply, so to obtain a boolean of whether a specific attribute is applied to a file or directory is done as follows:

```
Dim objFileInfo As New System.IO.FileInfo("c:\test.txt")
Dim bHidden As Boolean = ((objFileInfo.Attributes And
System.IO.FileAttributes.Hidden) <> 0)
```

Setting attributes can be done by assigning a value to the Attributes property:

```
Dim objDirInfo As New System.IO.DirectoryInfo("c:\testdir")
objDirInfo.Attributes = System.IO.FileAttributes.Hidden
```

The above code will hide the c:\testdir folder. Again, as with the SetAttributes() method, it is possible to set multiple attributes by separating each value with an Or operator:

```
Dim objDirInfo As New System.IO.DirectoryInfo("c:\testdir")
objDirInfo.Attributes = System.IO.FileAttributes.Hidden Or
System.IO.FileAttributes.ReadOnly
```

Maintaining the original attributes whilst applying new ones is also possible in the same fashion as with the SetAttributes() method:

```
Dim objDirInfo As New System.IO.DirectoryInfo("c:\testdir")
objDirInfo.Attributes = objDirInfo.Attributes Or
System.IO.FileAttributes.ReadOnly
```

The above code maintains the original attributes of the c:\testdir folder, but also sets it to be read-only.

Reading From and Writing To Text Files

The .NET Framework provides numerous classes and methods for both reading from and writing to files using a variety of different modes. This section introduces the simplest solutions to creating, reading from and writing to text files.

Reading From Files

The simplest way to read the entire contents of a file and store it in a string is by using the System.IO.File.OpenText() method. This method lets you specify the full path of the file to open and it returns a StreamReader object. The StreamReader has numerous methods, but the ReadToEnd() method returns the entire file as a string (assuming that it has been used on its own). The following code will store the contents of the file c:\test.txt in a string variable:

```
Dim objSR As System.IO.StreamReader =
System.IO.File.OpenText("c:\test.txt")
Dim strFile As String = objSR.ReadToEnd()
objSR.Close()
```

To read through a file, line-by-line, the ReadLine() method can be used as follows:

```
Dim objSR As System.IO.StreamReader =
System.IO.File.OpenText("c:\test.txt")
Dim strFile As String = ""
While objSR.Peek > -1
 strFile = strFile + objSR.ReadLine()
End While
objSR.Close()
```

The Peek() method reads the next character in the file, and returns -1 if it is the end of the file. The While loop ensures that all the lines in the file are read. It is also possible to read through the file, character by character using a similar technique:

```
Dim objSR As System.IO.StreamReader =
System.IO.File.OpenText("c:\test.txt")
Dim strFile As String
While objSR.Peek > -1
 strFile = strFile + Chr(objSR.Read())
End While
objSR.Close()
```

The Read() method returns the next character in the file as an Integer, so the Chr() function is used to convert the integer value to a character.

Writing To Files

The FileStream and StreamWriter classes are used when writing to text files. An instance of the FileStream class is required before a StreamWriter object can be created. The FileStream class can be created using its constructor, or by calling the System.IO.File.Open() method. Both methods require the name and path of the file and the file mode as a minimum to be specified. The file mode is a value from the System.IO.FileMode enumeration, listed in table 10.2.

Member	Description
Append	Appends any data written to the end of the file, or creates a new file if the file does not already exist.
Create	Creates a new file. If the file already exists, it is overwritten.
CreateNew	Creates a new file. If the file already exists, an exception is thrown.
Open	Opens a file.
OpenOrCreate	Opens a file, or creates a new file if the file does not already exist.
Truncate	Opens a file and immediately erases its content.

[Table 10.2 – System.IO.FileMode Enumeration Members]

The System.IO.File.OpenWrite() method also provides a shortcut for creating a FileStream that doesn't require a FileMode to be entered.

Before delving into actually writing to a file, let's first introduce the several methods of creating a StreamWriter class. The simplest is using the OpenWrite() method from the File class:

```
Dim objFS As System.IO.FileStream =
System.IO.File.OpenWrite("c:\test.txt")
Dim objSW As New System.IO.StreamWriter(objFS)
```

This code will create a FileStream object that can both be read and written to, and the specified file will be created if it doesn't already exist. The OpenWrite() method accepts that name of the file to open as a string and returns a FileStream object.

```
Dim objFS As System.IO.FileStream = System.IO.File.Open("c:\test.txt",
System.IO.FileMode.OpenOrCreate)
Dim objSW As New System.IO.StreamWriter(objFS)
```

The File class's Open() method has numerous overloaded versions, this being the simplest of them – the filename is passed as a string, and the mode to open the file in is specified. This is one of the values from the FileMode enumeration, and in this case is OpenOrCreate. Again, the method returns a FileStream object.

```
Dim objFS As System.IO.FileStream = System.IO.File.Open("c:\test.txt",
System.IO.FileMode.Append, System.IO.FileAccess.Write)
Dim objSW As New System.IO.StreamWriter(objFS)
```

This overloaded version of Open() accepts three parameters – the name of the file, the FileMode and a member from the FileAccess enumeration. The FileAccess enumeration has three members: Read, ReadWrite and Write. This specifies what operations can be performed on the resulting FileStream. The final overloaded version allows these three, and a FileShare enumeration member to be specified:

```
Dim objFS As System.IO.FileStream = System.IO.File.Open("c:\test.txt",
System.IO.FileMode.Append, System.IO.FileAccess.Write,
System.IO.FileShare.None)
Dim objSW As New System.IO.StreamWriter(objFS)
```

The FileShare enumeration has four members: None, Read, ReadWrite and Write. This specifies what level of control other streams have on the specified file after is has been opened. For example, if the FileShare.Read member is set, then other subsequent attempts to open and read the file whilst the original FileStream is still open will be allowed, but any attempts to write to the file by this second FileStream will fail.

The FileStream class can also be instantiated by itself, and does not have to be created by the File class's Open() method. Its constructor has numerous overloaded versions, of which the simplest can be used as follows:

```
Dim objFS As New System.IO.FileStream("c:\test.txt",
System.IO.FileMode.OpenCreate)
Dim objSW As New System.IO.StreamWriter(objFS)
```

The constructor allows the file to be passed as a string and the mode to open the file in as a member for the FileMode enumeration. Another version can be used to specify the FileAccess level as well:

```
Dim objFS As New System.IO.FileStream("c:\test.txt",
System.IO.FileMode.OpenCreate, System.IO.FileAccess.Write)
Dim objSW As New System.IO.StreamWriter(objFS)
```

In this case it will only be possible to write to the file, and not read from it.

The last version introduced here allows a FileShare value to be specified as well:

```
Dim objFS As New System.IO.FileStream("c:\test.txt",
System.IO.FileMode.OpenCreate, System.IO.FileAccess.Write,
System.IO.FileShare.Read)
Dim objSW As New System.IO.StreamWriter(objFS)
```

> Please take note that only a few versions of the FileShare constructor have been shown, and that the FileShare class contains much for functionality than this introduction to it demonstrates, so be sure to take a look at the .NET Framework documentation if you'd like to find out about some of the more advanced features of the FileShare class.

Once a StreamWriter object has been created, writing is a very simple affair. The two methods shown here involve writing to the file without automatically inserting line feeds, or writing to the file, one line at a time, where the CRLFs are inserted automatically. Both the Write() and WriteLine() methods of the StreamWriter feature numerous overloaded versions for writing most primitive data types to the file – from single characters (Char), to arrays of Chars, to strings, to Integers.

The following example opens a file, or creates new one if it doesn't already exist, and writes in the line "Hello World!" before closing it. The original content of the file is erased.

```
Dim objFS As System.IO.FileStream = System.IO.File.Open("c:\test.txt",
System.IO.FileMode.OpenOrCreate)
```

```
Dim objSW As New System.IO.StreamWriter(objFS)
objSW.WriteLine("Hello World!")
objSW.Close()
objFS.Close()
```

The same result could be achieved using the Write() method, although the CRLF has to be inserted manually:

```
Dim objFS As System.IO.FileStream = System.IO.File.Open("c:\test.txt",
System.IO.FileMode.OpenOrCreate)
Dim objSW As New System.IO.StreamWriter(objFS)
objSW.Write("Hello World!" & vbCrLf)
objSW.Close()
objFS.Close()
```

The following example will add a line to the file, without reading the previous contents and rewriting it, by using the Append FileMode:

```
Dim objFS As System.IO.FileStream = System.IO.File.Open("c:\test.txt",
System.IO.FileMode.Append, System.IO.FileAccess.Write)
Dim objSW As New System.IO.StreamWriter(objFS)
objSW.WriteLine("This is another line.")
objSW.Close()
objFS.Close()
```

Take note that in this example the FileAccess.Write member was explicitly specified when creating the FileStream. This is because the Append FileMode will only work when the file is opened is such a fashion. If it is not, an exception is raised.

Conclusion

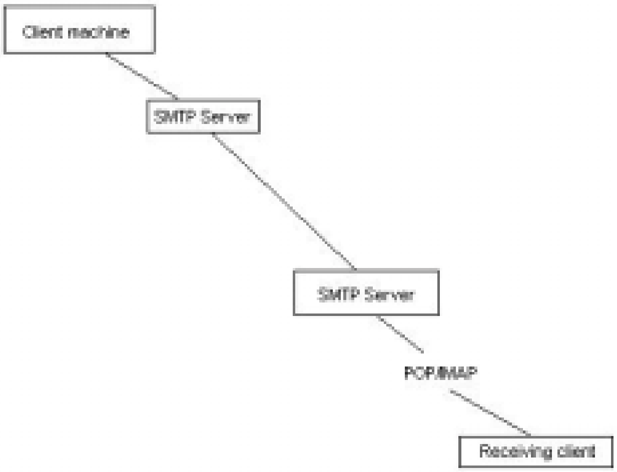
This chapter has given a brief introduction to some of the functionality offered by the System.IO namespace, which provides a very powerful framework for manipulating most aspects of the Windows filesystem. The filesystem can be a very useful method of storing data in certain scenarios, despite the power of modern RDBMS systems, and the System.IO namespace exposes numerous classes for making the task of manipulating directories, files and their contents fairly simple.

Chapter 11

Sending Email using ASP.NET

In order to understand how to send email in ASP.NET, a little background on sending email in general is required. Email is sent via the SMTP, or Simple Mail Transport, Protocol. There are two other high-level TCP/IP protocols involved in email, but I will come to those later. Basically the SMTP protocol defines how messages are sent over the Internet. This protocol is implemented in SMTP servers, whose purposes are to send and receive mail according to commands that clients issue to them. Generally users never communicate with SMTP servers directly – all communications are performed by a mail client such as Microsoft Outlook. However, in ASP.NET, you can’t use Outlook, so there are instances when you may need to communicate with the server directly.

Figure 11.1 shows a simplified version of how email is sent via SMTP servers.



[Figure 11.1 – A simple example of SMTP in action]

The SMTP server handles the actual scheduling and transfer of messages, so we don’t need to worry about the transfer of messages from one SMTP server to another. The area of the diagram that concerns us is sending the message from our machine to the SMTP server. However, IIS ships with an SMTP server of its own. This means that there is an SMTP server on the same machine as our web server. The .NET framework provides several classes that circumvent having to communicate with the IIS SMTP server directly, as they provide programmatic access to the server. I will elaborate later, but there is one drawback to this method – if it is not possible for the IIS SMTP server to be running, there is no way to specify another server to use.

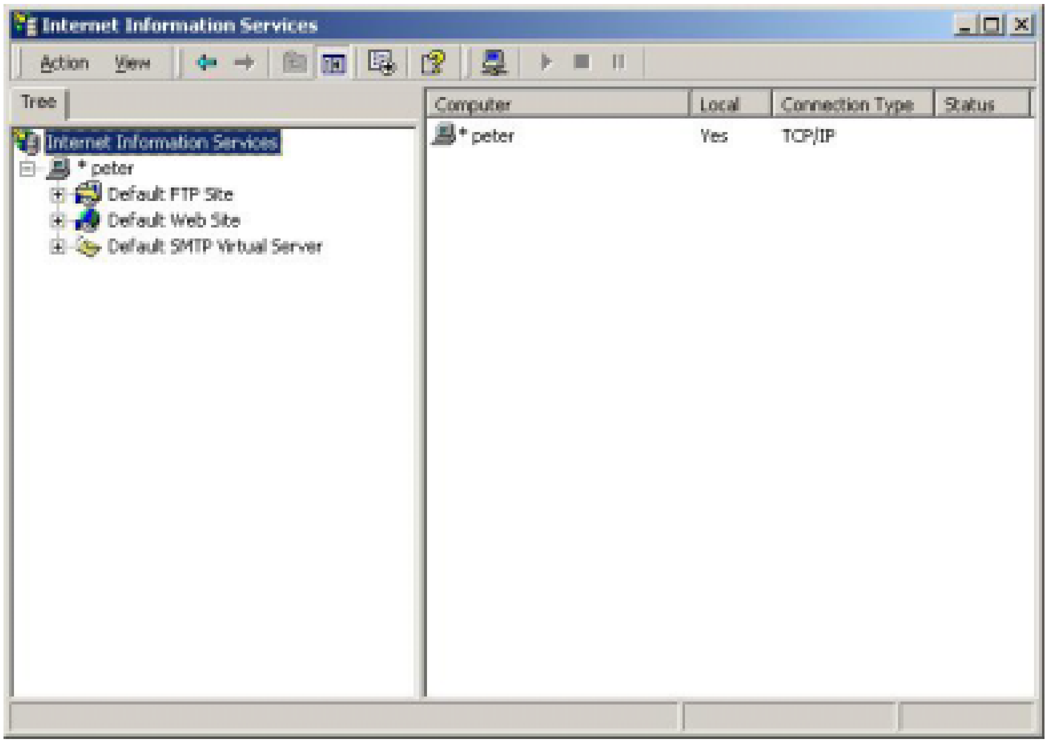
Sending Email in 1 Line

Yes, you read correctly. It’s possible to send an email in 1 line only. The .NET framework provides the MailMessage and SmtpMail classes for sending email, which together provide for most mail requirements, including attachments, but the SmtpMail class has a shared method “Send” that allows for a one-liner:

```
System.Web.Mail.SmtpMail.Send(From, To, Subject, Message)
```

All the parameters are Strings. However, as I’ve mentioned earlier, to test this out, IIS’s SMTP server needs to be running on the same machine as the application. To check this, and rectify it if possible, perform the following:

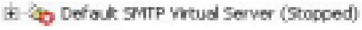
1. Click Start-Settings-Control Panel
2. Open “Administrative Tools”
3. Open “Internet Services Manager”
4. The MMC should show the following:


Internet Information Services
Action
View
Tree
Internet Information Services
* peter
Default FTP Site
Default Web Site
Default SMTP Virtual Server

Computer	Local	Connection Type	Status
* peter	Yes	TCP/IP	

[Figure 11.2 – IIS]

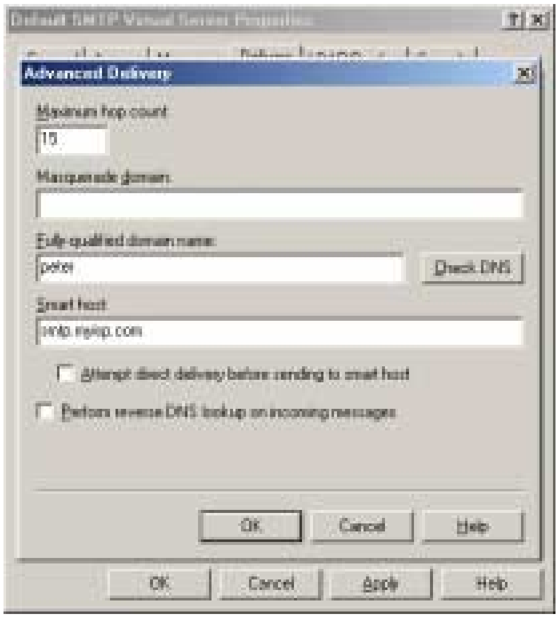
5. If the above is shown, then all is well, and you can close the MMC. If however, the “Default SMTP Virtual Server” node looks like this:



then you need to start your SMTP server. Do this by right-clicking the node and clicking “Start”. If an error occurs and the server does not start, the most probable cause is that you already have another application bound to port 25 (the port that SMTP servers run on by default). If this is the case, shut down your other server application, and try start IIS’s SMTP server again.

Depending on the situation, the SMTP server may require further configuration. If you are already on a network and know that your SMTP service is functioning correctly, then you may skip these steps. However, if you are either on a network that uses another machine as the SMTP server, or are not on a network at all, then you will need to configure your server to send all mail through another SMTP server, otherwise your machine will likely not be able to send mail to users on other systems and networks. In this case, perform the following steps:

1. Click Start-Settings-Control Panel
2. Open “Administrative Tools”
3. Open “Internet Services Manager”
4. Right-click the “SMTP Virtual Server” node and choose “Properties”.
5. Click on the “Delivery” tab, then on the “Advanced” button. A dialog similar to Figure 11.4 should appear.



Default SMTP Virtual Server Properties

Advanced Delivery

Maximum hop count: 15

Masquerade domain:

Fully-qualified domain name: peter

Smart host: smtp.myisp.com

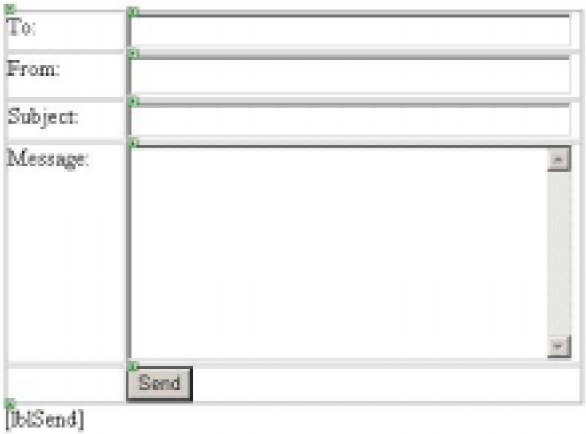
☐ Attempt direct delivery before sending to smart host

☐ Perform reverse DNS lookup on incoming messages

[Figure 11.4 – SMTP Delivery Advanced Settings]

6. Change the “Smart Host” field to the SMTP server that you would like to use to send mails. If you do not know what server to use for this purpose, refer to your mail client’s settings.

Now that the SMTP server is set up correctly, you’re ready to start sending email – in 1 line only. To demonstrate a real-world use of this, Figure 11.5 shows a web form in the Web Forms Designer, and Listing 11.1 shows the code that drives the web form.



[Figure 11.5 – Web Form UI for sending an email]

```
Public Class SimpleEmail
    Inherits System.Web.UI.Page
                Protected WithEvents lblSend As
System.Web.UI.WebControls.Label
    Protected WithEvents txtTo As System.Web.UI.WebControls.TextBox
    Protected WithEvents txtFrom As System.Web.UI.WebControls.TextBox
    Protected WithEvents txtSubject As System.Web.UI.WebControls.TextBox
    Protected WithEvents txtMessage As
System.Web.UI.WebControls.TextBox
    Protected WithEvents tblSimpleEmail As
System.Web.UI.HtmlControls.HtmlTable
    Protected WithEvents btnSend As System.Web.UI.WebControls.Button

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

    End Sub

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region
```

```
  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub

  Private Sub btnSend_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSend.Click
    System.Web.Mail.SmtpMail.Send(txtFrom.Text, txtTo.Text,
txtSubject.Text, txtMessage.Text)
    tblSimpleEmail.Visible = False
    lblSend.Text = "The email has been sent."
  End Sub
End Class
```

[Listing 11.1]

Only one line is used to actually send the email in this listing. It calls the shared Send method of the SmtpMail class, and passes 4 string parameters, each the contents of the appropriate TextBox web controls. Once the method has been called, the table containing the fields is hidden, and a message informing the user that the form has been sent is displayed. It would also be equally easy to omit a field, and have it filled in automatically in the code. The following line will send the message, but a “From” address set to “peter@mydomain.com” is automatically inserted, thus the txtFrom control can be deleted:

```
System.Web.Mail.SmtpMail.Send("peter@mydomain.com", txtTo.Text,
txtSubject.Text, txtMessage.Text)
```

It is possible to modify which SMTP server messages are sent via using the shared SmtpServer property of the SmtpMail class. This is a string property, which accepts the name of the machine that must be used as the SMTP Server. The local machine is assumed unless the property is explicitly set, which is why the above example does not set it.

Sending More Advanced Messages

Sending messages in only one line can be very convenient, however, there may often be times where the requirements for the message are more than simply specifying the to and from addresses and the subject and body of the message. Making use of carbon copies, blind carbon copies, attachments, and other features require the use of the MailMessage class.

The MailMessage class needs to be created as a new object. The appropriate properties of this object are then set, and the instance of the MailMessage class is passed to the shared Send method of the SmtpMail class, instead of the 4 string values as was previously demonstrated. Listing 11.2 shows the simple form example, with 3 extra fields (CC, BCC and Priority) that require the MailMessage class to be used, as the functionality offered by only the SmtpMail class is inadequate.

```
Public Class SimpleEmail
  Inherits System.Web.UI.Page
```

```
    Protected WithEvents tblSimpleEmail As
System.Web.UI.HtmlControls.HtmlTable
    Protected WithEvents lblSend As System.Web.UI.WebControls.Label
    Protected WithEvents txtTo As System.Web.UI.WebControls.TextBox
    Protected WithEvents txtFrom As System.Web.UI.WebControls.TextBox
    Protected WithEvents txtCC As System.Web.UI.WebControls.TextBox
    Protected WithEvents txtBCC As System.Web.UI.WebControls.TextBox
    Protected WithEvents lstPriority As
System.Web.UI.WebControls.DropDownList
    Protected WithEvents txtSubject As System.Web.UI.WebControls.TextBox
    Protected WithEvents txtMessage As
System.Web.UI.WebControls.TextBox
    Protected WithEvents btnSend As System.Web.UI.WebControls.Button

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

    End Sub

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
    End Sub

    Private Sub btnSend_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSend.Click
        Dim objMessage As New System.Web.Mail.MailMessage()
        objMessage.To = txtTo.Text
        objMessage.From = txtFrom.Text
        objMessage.Cc = txtCC.Text
        objMessage.Bcc = txtBCC.Text
        objMessage.Priority = CInt(lstPriority.SelectedItem.Value)
        objMessage.Subject = txtSubject.Text
        objMessage.Body = txtMessage.Text
        System.Web.Mail.SmtpMail.Send(objMessage)
        tblSimpleEmail.Visible = False
        lblSend.Text = "The email has been sent."
    End Sub
End Class
```

[Listing 11.2]

The CC and BCC are simply two additional text fields, but the Priority is implemented as a DropDownList web control in the user interface. The list has three options, “High”, “Low” and “Normal”, with the values “2”, “1” and “0” respectively.

The first line in the btnSend_Click event handler instantiates a new MailMessage object. The following 7 lines proceed to set the properties of the object with the appropriate values. The Priority property expects a value from the Mail.MailPriority enumeration, but it will accept Integer values (but not strings) that represent the values in the enumeration. It is also important to note that for properties that accept email addresses, such as To, CC and BCC, multiple addresses may be given by separating them with semi-colons. The line “System.Web.Mail.SmtpMail.Send(objMessage)” sends the message by passing the MailMessage object that was created to the Send method.

Send HTML-formatted Emails

HTML-formatted emails allow for more creative messages to be sent. If emails are being sent to inform the site administrator of events, then HTML-formatted emails needn’t be used (in fact, they’d probably just serve to annoy the site admin, since they use up unnecessary bandwidth), however, sending messages to customers using HTML can make them seem more exciting, and possibly include some corporate branding to appease the marketing masses. Whatever the reason, sending HTML formatted messages is simple. ASP.NET doesn’t offer as much control over the controls of such messages as mail clients such as Outlook Express do (for example, attached, embedded images are not supported), but most tasks are adequately performed. The primary area of importance is setting the message’s MailFormat to Html. Everything else is as it was in a normal message, with exception of the Body property, which will of course contain the HTML message (although it *can* contain only a plain text messages). Listing 11.3 shows the code to send an HTML message with an embedded image, taken from the website.

```
Dim objMessage As New System.Web.Mail.MailMessage()
objMessage.To = "peter@mydomain.com"
objMessage.From = "peter@mydomain.com"
objMessage.Subject = "This message is in HTML format!"
objMessage.BodyFormat = Mail.MailFormat.Html
objMessage.Body = "<html><head></head><body>Hi there<br /><br
/><b>Here's an embedded image:</b><br /><img
```

```
src=""http://www.mydomain.com/images/nicepicture.jpg""><br /><br
/>Enjoy!</body></html>"
System.Web.Mail.SmtpMail.Send(objMessage)
```

The BodyFormat property expects a value from the MailFormat enumeration, which has the options either “Plain” or “Html”. The complexity of the HTML used in the message is totally dependant on the receiver’s email reader. Outlook Express, for example, uses the Internet Explorer rendering engine to render HTML email messages, and thus the current IE versions engine will be used to this end. However, other mail readers may only have support for HTML 3.2. It is also important to note that while HTML messages are useful, they should not be overused, and should not contain too many large images, lest dial-up users become annoyed with message load time.

Sending Attachments

Sending attachments is achieved through the use of the MailAttachment class, in conjunction with the Attachments property of the MailMessage class, which is of type IList, the class from which the likes of the Array class, and numerous collection classes for web controls are derived. The Add method of the IList class allows instantiated MailAttachment classes to be added to the list of attachments for a particular message. The MailAttachment class contains properties for the filename of the file to be attached on the server, and the encoding of the file. The constructor for this class allows either the filename (of type string) or the filename and the appropriate encoding to be set when the class is instantiated. Listing 11.4 shows the code to attach a file on the server to a message and send it.

```
Dim objMessage As New System.Web.Mail.MailMessage()
Dim objAttachment As New
System.Web.Mail.MailAttachment("c:\file_to_attach.txt")
objMessage.Attachments.Add(objAttachment)
objMessage.To = "peter@mydomain.com"
objMessage.From = "peter@mydomain.com"
objMessage.Subject = "This message has an attachment!"
objMessage.Body = "This message has an attachment."
System.Web.Mail.SmtpMail.Send(objMessage)
```

[Listing 11.4]

Multiple Attachments

In order for multiple attachments to be sent, multiple MailAttachment classes must be created. In this case, it may be useful to create an array of MailAttachments, and then loop through the array, adding the classes to the Attachments IList of the MailMessage class. Listing 11.5 shows the code to add 3 attachments using this method.

```
Dim objMessage As New System.Web.Mail.MailMessage()
Dim objAttachments(2) As System.Web.Mail.MailAttachment
Dim i As Integer
```

```
objAttachments(0) = New
System.Web.Mail.MailAttachment("c:\attachment1.txt")
objAttachments(1) = New
System.Web.Mail.MailAttachment("c:\attachment2.txt")
objAttachments(2) = New
System.Web.Mail.MailAttachment("c:\attachment3.txt")
For i = 0 To UBound(objAttachments)
 objMessage.Attachments.Add(objAttachments(i))
Next
objMessage.To = "peter@mydomain.com"
objMessage.From = "peter@mydomain.com"
objMessage.Subject = "This message has several attachments!"
objMessage.Body = "This message has several attachments."
System.Web.Mail.SmtpMail.Send(objMessage)
```

[Listing 11.5]

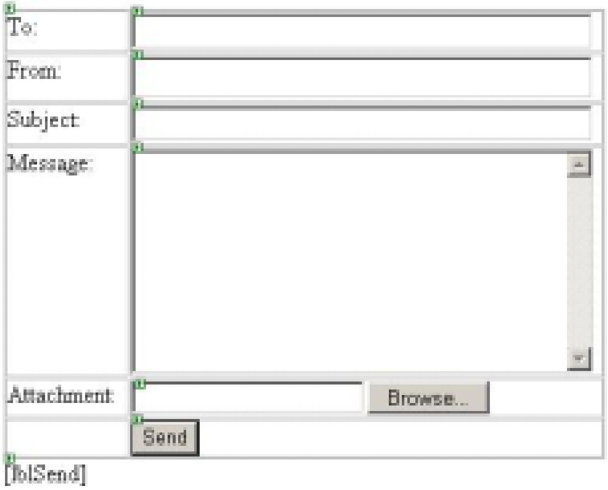
Since array subscripts always begin at zero, dimensioning objAttachments(2) actually creates 3 elements. This line creates an array, from 0 to 2, of type MailAttachment. However, although the elements are of type MailAttachment they do not as yet contain instances of MailAttachment classes, which explains the need for the respective New statements in the following lines. The loop is from zero to the upper bound of the array. This effectively loops through all the elements in the given array, adding each element (which is a MailAttachment object) to the Attachments IList. The rest of the MailMessage is set up, and sent as per normal.

File Uploading and Attachments

Thanks largely to ASP.NET's support of file uploading without the need for 3rd party components, as in previous versions, using file uploading in conjunction with attachments is now a trivial task, compared to what may have been daunting before the release of ASP.NET, and only involves a few more lines of code that simply sending an ordinary message.

Figure 11.6 shows a web form similar to that in Figure 11.5, which has been revised to include an HtmlInputFile control.

When performing HTTP file uploads, it is important that the form's enctype property is set to "multipart/form-data". If this is not done, the form upload will not execute correctly.



[Figure 11.6 – Web Form with HTML File Upload control]

The click event handler of the Send button will first save the uploaded file to the server’s hard drive. It will then create a MailAttachment class with the file, and proceed to continue with the MailMessage class normally. Listing 11.6 shows the entire source for the codebehind for this web form.

```
Public Class SimpleEmail
   Inherits System.Web.UI.Page
   Protected WithEvents tblSimpleEmail As
System.Web.UI.HtmlControls.HtmlTable
   Protected WithEvents lblSend As System.Web.UI.WebControls.Label
   Protected WithEvents txtTo As System.Web.UI.WebControls.TextBox
   Protected WithEvents txtFrom As System.Web.UI.WebControls.TextBox
   Protected WithEvents txtSubject As System.Web.UI.WebControls.TextBox
   Protected WithEvents txtMessage As
System.Web.UI.WebControls.TextBox
   Protected WithEvents filAttachment As
System.Web.UI.HtmlControls.HtmlInputFile
   Protected WithEvents btnSend As System.Web.UI.WebControls.Button

#Region " Web Form Designer Generated Code "

   'This call is required by the Web Form Designer.
   <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

   End Sub

   Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
```

```
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
    End Sub

    Private Function GetFileName(ByVal Path As String) As String
        Return Right(Path, Len(Path) - InStrRev(Path, "\"))
    End Function

    Private Sub btnSend_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSend.Click
        filAttachment.PostedFile.SaveAs("c:\temp\" +
GetFileName(filAttachment.PostedFile.FileName))
        Dim objAttachment As New System.Web.Mail.MailAttachment("c:\temp\"
+ GetFileName(filAttachment.PostedFile.FileName))
        Dim objMessage As New System.Web.Mail.MailMessage()
        objMessage.Attachments.Add(objAttachment)
        objMessage.To = txtTo.Text
        objMessage.From = txtFrom.Text
        objMessage.Subject = txtSubject.Text
        objMessage.Body = txtMessage.Text
        System.Web.Mail.SmtpMail.Send(objMessage)
        tblSimpleEmail.Visible = False
        lblSend.Text = "The email has been sent."
    End Sub
End Class
```

[Listing 11.6]

The call of the HttpPostedFile class's SaveAs method requires particular attention. The FileName property of the HtmlPostedFile class (the type of the PostedFile property of the HtmlInputFile class) contains the full physical path of the uploaded file on the *client*. Thus, if the client uploaded the file from "C:\My Documents\Messages\Attachment.doc", the value of this FileName property would be that entire path. It is obviously unsuitable to have the entire path used when storing the file on the server, in which case the server would have files liberally scattered throughout its hard drive. However, it would be desirable to keep the original name of the file for descriptive purposes when attaching it to the message. The GetFileName function receives a full physical path as a string parameter and returns only the filename and extension. It does this by searching for the last backslash, and copying the remainder of the full path after it. Thus, the SaveAs call saves the uploaded file to “C:\Temp\<filename>.<ext>”. In the following line, where the MailAttachment is created, this path is then used again. The rest of the code remains the same as for a normal mail message with a normal attachment.

Mass Mailing

There are two primary methods of mass mailing, each with its respective advantages and disadvantages. The first is sending one message, with all the recipients in the blind carbon copy field. This message has the advantage of only requiring one message to be sent (through your server, at least), thus reducing bandwidth wastage, but carries the disadvantage that the message cannot be customised for each recipient. The second method is to create a new mail message for each recipient. This has the disadvantage of generally creating more work for your SMTP server, but has the advantage of being able to tailor the message for each recipient.

BCC Mass Mailing

Listing 11.7 demonstrates how to send a message to all the email addresses in a string array, using the BCC field. Note that the message sent to each recipient is the same, but that only one MailMessage object is created.

```
Dim objMessage As New System.Web.Mail.MailMessage()
Dim strRecipients(4) As String
strRecipients(0) = "peter@mcmahon.com"
strRecipients(1) = "leon@cilliers.com"
strRecipients(2) = "leonard@fienberg.com"
strRecipients(3) = "lionel@guimaraes.com"
strRecipients(4) = "craig@macintosh.com"
objMessage.To = "webmaster@mysite.com"
objMessage.From = "webmaster@mysite.com"
objMessage.Bcc = Join(strRecipients, ";")
objMessage.Subject = "This is a mass mailing!"
objMessage.Body = "This message is part of a mass mailing."
System.Web.Mail.SmtpMail.Send(objMessage)
```

[Listing 11.7]

The string array strRecipients is used to hold the email addresses of 5 recipients. Although the addresses are hard-coded in this example, they could just have easily been retrieved from a mailing database. The BCC property of the MailMessage class is set to a semi-colon separated list of email addresses. The Join function takes an array as its first parameter, and a string delimiter as its second. This function returns one string that contains each element of the array, separated by the specified delimiter, which is in this case a semi-colon.

Individual Message Mass Mailing

Listing 11.8 shows an alternative method of mass mailing that sends an individual message to each recipient by looping the creation and sending of multiple messages.

```
Dim objMessage As System.Web.Mail.MailMessage
Dim strRecipients(4) As String
Dim i As Integer
strRecipients(0) = "peter@mcmahon.com"
strRecipients(1) = "leon@cilliers.com"
strRecipients(2) = "leonard@fienberg.com"
strRecipients(3) = "lionel@guimaraes.com"
strRecipients(4) = "craig@macintosh.com"
For i = 0 To UBound(strRecipients)
  objMessage = New System.Web.Mail.MailMessage()
  objMessage.To = strRecipients(i)
  objMessage.From = "webmaster@mysite.com"
  objMessage.Subject = "This is a mass mailing!"
  objMessage.Body = "Hi " + strRecipients(i) + ", this message was generated
especially for you!"
  System.Web.Mail.SmtpMail.Send(objMessage)
Next
```

Again, strRecipients is a hard-coded array of recipients. However, instead of creating one MailMessage object and assigning a semi-colon separated string of address to a recipient field, a For loop construction loops through all the recipients and generates a separate message for each recipient. Note that the message body is different for each recipient. Naturally if the email addresses were being acquired from a database rather than a string array, the message could include the recipient’s name and other details, depending on how much information the mailing database contained.

Conclusion

This chapter has demonstrated some of the most important features of the System.Web.Mail namespace in ASP.NET, which allows you to perform most mailing operations simply and easily. The chapter covered setting up the SMTP server is IIS, sending an email in one line only, HTML messages, attachments, user upload attachments in addition to 2 commonly used techniques for mass mailing.

Chapter 13

Creating your own User Controls

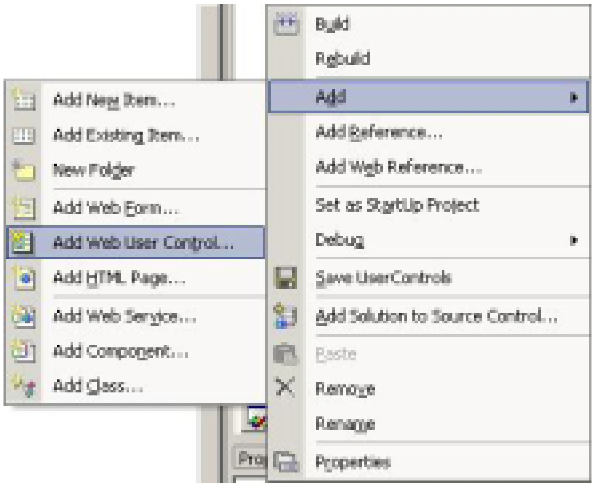
The HTML and Web controls offered by the .NET Framework class library cover a wide variety of generic functionality required to build web applications, from labels, to text inputs, to validation, to data grids. However, the framework naturally cannot provide for the needs of all applications, and in-keeping with the principals of component-oriented programming to encapsulate code and encourage code reuse, ASP.NET allows you to build your own custom controls for use on web forms. User Controls are very powerful, but also easy to build and use. All custom controls are inserted very similarly to the way in which the standard HTML and Web controls are inserted, and Visual Studio.NET provides drag-and-drop functionality for user-built controls as well. Since VB.NET supports object oriented programming, and in particular implementation inheritance, it is possible not only to build your own control from scratch, but also to derive from one of the HTML or Web controls, and build upon it.

User Controls provide a great way to make the user interface of your web application more modular in design, which in itself brings many benefits. ASP programmers may have wondered what improvement ASP.NET has made to server-side includes. The simple answer is the introduction of user controls. Server-side includes allowed other files (typically ASP pages) to be inserted at specified points throughout an ASP page. SSIs were commonly used for modularizing the design of ASP pages for creating maintainable basic designs for all pages to be based upon by allowing common aspects of the design (such as the header and footer) of a page to be stored in separate files. This makes a significant improvement to how easy a site is to maintain by allowing changes in one file (e.g. the footer) to be automatically applied to all the files that “include” the footer. This is the same principal as using external CSS stylesheets, such that a change of a color in the actual stylesheet file affects the entire site’s colors. User Controls are best used in web applications to achieve the same kind of maintainability. However, since user controls are classes, they offer some major benefits over server-side includes, such as the ability to expose properties and methods.

Getting Started

There are essentially two methods of creating user controls – one where the control’s user interface is specified in an .ascx file, with its logic implemented in a code-behind file, and the second where a class manually creates the control through code. This chapter will discuss and delve into the advantages and disadvantages of both methods, but the former is certainly the simplest, and the one that the VS.NET environment offers support for.

To begin with, create a new web project. Once the project has been created, right click on the project node in the Solution Explorer and select Add-Add Web User Control, as shown in Figure 13.1, to add a new user control, “Footer.ascx”.



[Figure 13.1 – Adding a new user control]

Once this has been done, a new page should appear in the IDE. This will look almost identical to a new web form, except the filename in the tabs will indicate that the extension is .ascx, and not .aspx, denoting the file as a web user control. This control will simply be used to append the current date to pages, so add a label to the "page", just as you would for a web form and name it, "lblDate". Switch to the "Code View", either by right-clicking the "page" and choosing the appropriate option from the context menu, or by doing so in the Solution Explorer, as you would for a normal web form. Listing 13.1 shows what the code should look like.

```
Public MustInherit Class Footer
    Inherits System.Web.UI.UserControl
    Protected WithEvents lblDate As System.Web.UI.WebControls.Label

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

    End Sub

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region
```

```
    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load

    End Sub

 End Class
```

[Listing 13.1]

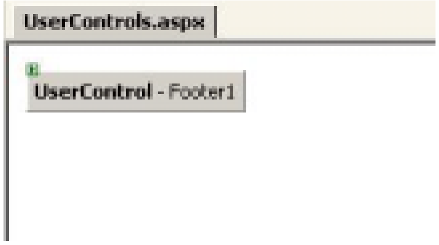
First up, it’s clear that the control is implemented as a class that inherits from the System.Web.UI.UserControl class. This class provides the basic functionality for a user control. As with ASP.NET pages, the next section in the code-behind file is the list of user-added members. In this case, the only control that has been added is the label. The two subs in the “Web Form Designer Generated Code” region deal with initializing components that may have been added to the form at design time. Finally, the Page_Load event handler is available as a place to put custom initialization code. Put the following line into this sub:

```
lblDate.Text = Now.ToLongDateString()
```

This will set the Text property of the label to the current date in the long date format when the user control is loaded.

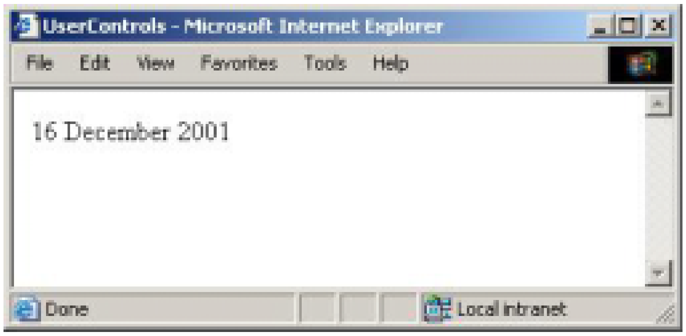
Using the Control

Once the control has been created, using it on a web form is a trivial affair. The simplest way to add a user control to a form is by dragging the user control .ascx file node in the Solution Explorer onto the form, in exactly the same way as controls are added from the toolbox by dragging the control from the toolbox onto the form. Figure 13.2 shows the Footer control on a new web form.



[Figure 13.2 – A blank web form with the footer control added]

When this page is viewed in the browser, the code in the user control will be executed, resulting in the date being displayed, as shown in figure 13.3.



16 December 2001

[Figure 13.3]

User Controls can, however, consist of complex arrangements of web controls, not just a single label as with the above example. It is also possible to dynamically add web controls to a user control at runtime, so that user controls can be used for displaying data-driven menus, for example. However, no matter how complex or simple a user control is, using it in a web form is always as simple as dragging and dropping the control from the Solution Explorer.

When a user control is added to a form, several lines of code are added to the user interface file. These lines tell the ASP.NET compiler where it must load the control from (i.e. either the location and name of the .ascx file, or the name of the control's assembly and namespace), and then where the control must be inserted in the page. Inspecting the HTML of the web form in the above example will reveal the code in listing 13.2.

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="UserControls.aspx.vb"
Inherits="UserControls.UserControls"%>
<%@ Register TagPrefix="uc1" TagName="Footer" Src="Footer.ascx" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
 <HEAD>
  <title>UserControls</title>
  <meta content="Microsoft Visual Studio.NET 7.0"
name=GENERATOR>
  <meta content="Visual Basic 7.0" name=CODE_LANGUAGE>
  <meta content=JavaScript name=vs_defaultClientScript>
  <meta content=http://schemas.microsoft.com/intellisense/ie5
name=vs_targetSchema>
 </HEAD>
<body>
 <form id=Form1 method=post runat="server">
  <uc1:Footer id=Footer1 runat="server"></uc1:Footer>
 </form>
</body>
</HTML>
```

[Listing 13.2]

The @ Register directive is used to define the location of a user control that can be used in a page, as well as the tags that are used to identify the instances of the control. It has two forms – one that allows the user control filename to be specified, and another allowing the assembly and namespace to be specified. Visual Studio inserts the former:

```
<%@ Register TagPrefix="uc1" TagName="Footer" Src="Footer.ascx" %>
```

The TagPrefix and TagName attributes specify what the tags that will be used to insert the control will look like. When inserting a regular Web Control, the "tag prefix" is always "asp", and the "tag name" depends on the control – for example, "TextBox". The Src attribute specifies the user control file.

This directive only defines how "Footer" controls can be added, but does not actually add any itself. This is done using tags similar to those for Web Controls, but using the parameters specified in the @ Register directive. In this case, the tag for inserting the "Footer" control is <uc1:Footer>, because the tag prefix is "uc1" and the tag name is "Footer". As with all Web Controls, user controls must also include a runat attribute, and an id attribute is normally given so that the control can be programmed against. The final tag to insert the control in the example is:

```
<uc1:Footer id=Footer1 runat="server"></uc1:Footer>
```

The generic tag prefixes that Visual Studio creates are not very descriptive, so it may be useful to change them. To do so, simply edit the TagPrefix attribute of the @ Register directive, and suitable modify the tags of all the user controls on the form.

Creating Controls with Properties

Once of the major pitfalls of using server-side includes in the past was that passing "property values" to them was difficult at best, since they weren't designed for use as objects on a page. Since user controls are fully-fledged classes, they can expose, amongst other things, properties. Properties can be used in a wide variety of situations to greatly increase the functionality of user controls. They can perform simple tasks, such as providing the color for a control, to as relatively complex as specifying the type and location of a data store for a custom data display control. However, one of the major advantages of using properties is their ease of use – they can be specified inside the user control tag, just as with regular Web Controls, although it is still possible to manipulate them through server-side code.

One of the most common tasks of properties in user controls is to directly manipulate the contents and properties of the user control. Most websites try to achieve continuity by using a consistent color scheme and page layout. This often leads to the development of standard headers, footers and menu bars for all the pages in the site, or for each major section in it. User Controls are great for developing such elements, as not only is creating the design for a page a matter of inserting two or three user controls, but if the design or contents of the header, for example, needs to be changed, the change need only be applied to the header user control file. Properties are particularly useful for an element such as the header, which may contain the name of the current section, or perhaps the page title within it. Without properties, it would not be possible for the same user control to display different text in different instances. Using properties, it would for example, also be possible for a section of the header to be colored according to the section that the user is in, for example, gray for products, light blue for company information etc. Properties would also have great application in a menu control, where a property could specify which item should be highlighted.

To illustrate the use of properties, the following example will create a header control for a website. The header will feature a title that is constant, and beneath it, right-aligned text containing the name of the section that the user is in. The control will also allow the color of the “section” bar to be specified. The control will be create using a table with two rows, with one cell in each row.

Firstly, in order for the color of the table rows to be manipulated programmatically, a server-side control must be used. Either the Table Web Control or Table Html Control can be used. Secondly, for the section name to be inserted, a Label control (again, either Html or Web Control) must be used. Listing 13.3 shows the HTML of the UI for the header controls.

```
<%@ Control Language="vb" AutoEventWireup="false"
Codebehind="Header.ascx.vb" Inherits="UserControls.Header"
TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<table border="0" cellpadding="0" cellspacing="0" width="100%"
id="tblHeader" runat="server">
        <tr>
                <td bgcolor="#000066">
                        <font face="Arial" size="5" color="#ffffff">XYZ
Inc.</font>
                </td>
        </tr>
        <tr>
                <td align="right">
                        <font face="Arial" size="2"
color="#000000"><asp:Label ID="lblSection"
runat="server"></asp:Label></font>
                </td>
        </tr>
</table>
```

[Listing 13.3]

The top row’s cell has a dark blue background color, and contains large white text with the name of the site, “XYZ Inc.” The second row’s cell does not have a background color specified, as this will be set by a user control property. There is no text in this cell yet, but a Label Web Control is present, which will later contain the section name.

The control’s server-side code is where the property definitions occur. For this control, two private variables will be used to store the row color and the section name internally. These will be declared in the user control class as follows:

```
Private colBarColor As Color
Private strSection As String
```

The properties will have a public scope, as they must be available to the web form class. Both will simply set their respective private variable when they are assigned a value, and return the private variable value when requested, so the definitions for the two properties are:

```
Public Property BarColor() As Color
  Get
```

```
    Return colBarColor
  End Get
  Set(ByVal Value As Color)
    colBarColor = Value
  End Set
End Property

Public Property Section() As String
  Get
    Return strSection
  End Get
  Set(ByVal Value As String)
    strSection = Value
  End Set
End Property
```

> Since the property get and set accessors are doing nothing other than assigning and returning the value of a private variable, it would be possible to expose public variables as "properties" of the user control instead of real properties in this instance, so both Property blocks and the private variables could be scrapped, and replaced by two public variables, BarColor and Section, of type Color and String respectively.

All public properties (and variables) can be set as properties in the user control's tag on the web form using the name of the property as the attribute in the tag. For this header control, setting the BarColor and Section properties could be done as follows:

```
<XYZInc:Header BarColor="propertyvalue" Section="propertyvalue"
runat="server">
```

It is important to keep this in mind when choosing the names of properties, so that the user does not have to specify attributes with names such as "strSection".

The UserControl class exposes a Render() method that is called when the control must be displayed. In order to use the BarColor and Section properties that have been specified by the user, this method should be overridden, so that the appropriate properties of the table and label can be set before they are rendered:

```
Protected Overrides Sub Render(ByVal output As HtmlTextWriter)
  tblHeader.Rows(1).Cells(0).BgColor =
ColorTranslator.ToHtml(BarColor)
  lblSection.Text = Section
  RenderChildren(output)
End Sub
```

Since the BgColor property of the HtmlTableCell class accepts a String, the BarColor property (which is of type System.Drawing.Color) must first be converted. The ColorTranslator's ToHtml method accepts a Color value and returns the HTML equivalent. The Section property, which is a String, is directly assigned to the label's

Text property, without any manipulation. Finally the RenderChildren() method is called, which writes the actual Html for the controls contained in the user control.

To use the control, it must first be added to a web form. The easiest method of doing this is by dragging-and-dropping the .ascx file from the Solution Explorer onto the form. Once this has been done, switch to the HTML view and location the User Control tag. It will look something like this:

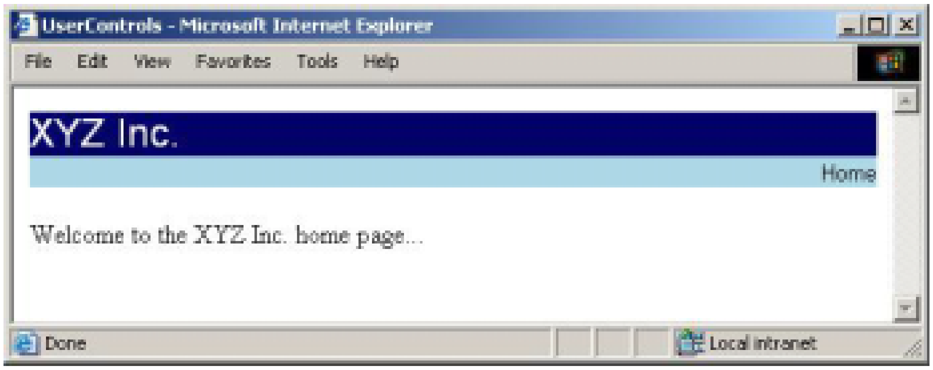
```
<uc1:Header id="Header1" runat="server"></uc1:Header>
```

The control expects two properties to be specified – BarColor and Section. Both are specified using attributes of the user control tag, as follows:

```
<uc1:Header id="Header1" BarColor="LightBlue" Section="Home"
runat="server"></uc1:Header>
```

> When designing user controls, it is important to provide for situations where the user does not specify values for properties. This is normally best done assigning default values for all the properties using initializers on the private variables storing their values, so that even if the user does not specify a value for any given property, the control will still function correctly.

Once these attributes have been assigned values, the rest of the page can be built. Figure 13.4 shows a web form loaded in the browser using the header control.



XYZ Inc.

Home

Welcome to the XYZ Inc. home page...

[Figure 13.4]

Programmatically Assigning User Control Properties

Web Controls can be manipulated via server-side code in the code-behind file of ASP.NET pages. This has many advantages, and makes it possible to build dynamic controls that can modify their appearance and behaviour through server-side property manipulation. User controls also have this ability, and there is no difference between assigning user control properties or web control properties. However, before properties can be programmatically assigned (or indeed any of the members of the user control can be accessed in the code-behind file), a declaration for the control must manually be entered.

When a web control is added to a page, Visual Studio.NET automatically inserts a declaration for the control in the page’s code-behind file. For example, if a TextBox web control with the name, txtAge is entered, the following declaration will appear in the code-behind file:

```
Protected WithEvents txtAge As System.Web.UI.WebControls.TextBox
```

However, Visual Studio does not automatically enter this declaration for user controls, so they must be manually entered if they are to be accessible in the server-side code. For the Header control that was added in the above example, the name of the control is Header1, so the declaration that must be added for it is:

```
Protected WithEvents Header1 As Header
```

The properties of the Header1 control can now be manipulated programmatically, just as with any other Html or Web Control. For example, the following event handler for the click event of a Button control will result in the Header's bar changing color when a btnChangeColor Button is clicked:

```
Private Sub btnChangeColor_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnChangeColor.Click
    Header1.BarColor = Color.Bisque
End Sub
```

Of course, for the Header control, programmatically modifying properties doesn't yield many possibilities, but in the next section dealing with exposing methods, properties become significantly more important.

Exposing Methods in a User Control

Most Web Controls expose methods allowing the page to manipulate it. For example, the validator controls expose a Validate() method, and the data controls expose a DataBind() method. To demonstrate how to use properties and methods together to build rich controls, this section will create a control that displays a list, based on the items added using methods, and with visual aspects controlled by properties.

Firstly, the control will expose several properties controlling the appearance of the list. The list will allow one item to be "selected", so there must be a property that specifies which item is selected. Secondly, two properties must be available to specify what color selected items and unselected items must be displayed using. Thirdly, a property must be available allowing the user to programmatically check the contents of any item on the list. This property must be read-only. Finally, another read-only property by be defined that allows the user to obtain the count of how many items are in the list.

All the properties will use private variable to store their values internally. One more private variable must also be declared which will store the actual list (which also stores the Count property's value). For this purpose, an ArrayList class is perfect. The private variable definitions are as follows:

```
Private Items As New ArrayList()
Private intSelectedItem As Integer = -1
Private colSelectedColor As Color = Color.Gray
Private colUnselectedColor As Color = Color.White
```

The ArrayList class allows a list of Objects to be stored, and accessed through an Items property, using an index. The intSelectedItem variable is initialized to –1 so that by default no items are selected. The default colors for selected and unselected items are gray and white respectively.

The following are the actual property definitions:

```
Public ReadOnly Property Item(ByVal index As Integer) As String
  Get
    Return CStr(Items.Item(index))
  End Get
End Property

Public ReadOnly Property Count() As Integer
  Get
    Return Items.Count
  End Get
End Property

Public Property SelectedItem() As Integer
  Get
    Return intSelectedItem
  End Get
  Set(ByVal Value As Integer)
    If Value <= Items.Count - 1 Then
      intSelectedItem = Value
    Else
      intSelectedItem = Items.Count - 1
    End If
  End Set
End Property

Public Property SelectedColor() As Color
  Get
    Return colSelectedColor
  End Get
  Set(ByVal Value As Color)
    colSelectedColor = Value
  End Set
End Property

Public Property UnselectedColor() As Color
  Get
    Return colUnselectedColor
  End Get
  Set(ByVal Value As Color)
    colUnselectedColor = Value
  End Set
End Property
```

The Item() property is read-only, so only the Get accessor is included. The property is of type String, as even though the ArrayList can store objects of all types, only Strings will be stored in this user control, thus the value is cast as a string using CStr() before being returned. The Count property is simply a wrapper for the ArrayList's Count property, since the user doesn't have direct access to the Items object. The SelectedItem Set accessor uses an If condition to ensure that the SelectedItem is never larger than the total number of items in the list (although it can be negative, indicating that nothing is selected). The remaining two properties simply access or store values to their respective private variables.

The user control will expose two methods – one for adding items to the list, and one for removing items. The AddItem() method will accept a string as a parameter, which will be added to the Items ArrayList object. The RemoveItem() method will accept an Integer representing the index of the item to remove from the Items ArrayList. It performs some simple bounds checking to ensure that the parameter passed to it is within range before attempting to remove the item. The implementation for these two methods follows:

```
Public Sub AddItem(ByVal item As String)
  Items.Add(item)
End Sub


Public Sub RemoveItem(ByVal index As Integer)
  If (index >= 0) And (index <= Items.Count - 1) Then
    Items.RemoveAt(index)
  End If
End Sub
```

Theses methods and properties will be able to control all aspects of the control. However, before the code to render the control is written, the following UI code must be placed in the control's user interface file:

```
<table border="0" id="tblSelectionTable" runat="server">
</table>
```

This table will be used to display the items in the ArrayList. It is important to note that this code could have been replaced by creating an HtmlTable control in the server-side code.

When the ASP.NET engine creates an instance of a user control, several methods are called within the control. The Control class's CreateChildControls() method is called when the control must create its child controls and add them to the control tree prior to rendering. The method must be overridden so that our table list control can create the actual list in HTML from the ArrayList object. Listing 13.4 shows the overridden method.

```
Protected Overrides Sub CreateChildControls()
  Dim i As Integer
  For i = 0 To Items.Count - 1
    Dim tr As New HtmlTableRow()
    Dim tc As New HtmlTableCell()
```

```
      tc.InnerHtml = CStr(Items.Item(i))
      If SelectedItem = i Then
        tc.Attributes.Add("bgcolor", ColorTranslator.ToHtml(SelectedColor))
      Else
        tc.Attributes.Add("bgcolor",
  ColorTranslator.ToHtml(UnselectedColor))
      End If
      tr.Cells.Add(tc)
      tblSelectionTable.Rows.Add(tr)
    Next
  End Sub
```

[Listing 13.4]

The objective of this method is to create a single row for each item in the Items ArrayList. If the item is “selected”, then that background color for the cell must be the SelectedColor property, and if not, the UnselectedColor property must be used.

The first two lines declare a counter variable and start a For loop through all the items in the ArrayList. Next, a table row and table cell are declared. The table cell is then set to contain the text from the corresponding item in the Items ArrayList. An If condition checks to see whether the current item is “selected”. If so, the cell’s bgcolor attribute is set to the HTML color value for the SelectedColor property, and if not, the UnselectedColor property is used. Finally, the cell is added to the row, and the row is added to the table.

Using the “SelectionTable” Control

The SelectionTable control can be added to a web form the same way in which any other control is added. Attributes will be used to specify the SelectedColor and UnselectedColor properties, so the control’s tag may look like this:

```
  <uc1:SelectionTable id="SelectionTable1" SelectedColor="LightBlue"
  UnselectedColor="Gray" runat="server"></uc1:SelectionTable>
```

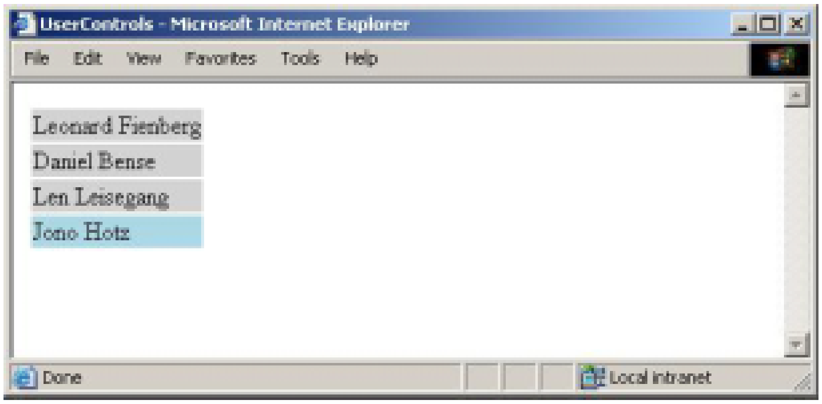
The declaration for the control must be made in the web form’s code behind file. Assuming that the control is named “SelectionTable1”, the declaration should be written as follows:

```
  Protected WithEvents SelectionTable1 As SelectionTable
```

The Page_Load event handler will be used to fill the SelectionTable control with values using the AddItems() method:

```
  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
  System.EventArgs) Handles MyBase.Load
    SelectionTable1.AddItem("Leonard Fienberg")
    SelectionTable1.AddItem("Daniel Bense")
    SelectionTable1.AddItem("Patrick Collins")
    SelectionTable1.AddItem("Len Leisegang")
    SelectionTable1.AddItem("Jono Hotz")
    SelectionTable1.RemoveItem(2)
    SelectionTable1.SelectedItem = 3
  End Sub
```

This SelectionTable will contain a list of names. The RemoveItem() method is used to remove the 3rd item (remember, the ArrayList’s index is zero-based, so the first item’s index is 0), and the 4th item will be selected. Figure 13.5 shows the result in a browser.



Leonard Fienberg
Daniel Bense
Len Leisegang
Jono Hotz

[Figure 13.5]

To demonstrate the use of the SelectionTable’s Item() property, place a label on the form, and add the following line to the Page_Load event handler:

```
  Label1.Text = SelectionTable1.Item(0)
```

When the page is loaded, the label should display, “Leonard Fienberg”.

Storing State in User Controls

Storing State using the StateBag

The SelectionTable control created in the previous section has a major problem – the state of the control is not stored. This means that every time a postback occurs (i.e. the page containing the control reloads), the control’s contents and properties are all lost. This is obviously an unacceptable pitfall, as most ASP.NET pages rely on postbacks for their functionality. What needs to happen, is that the control must “remember” what items it stored, and what its property values were, so that when a postback occurs and a the control is re-created, it can simply start up where it left off. Fortunately the Control class provides a ViewState property, which is a reference to a StateBag class for precisely this purpose. Incidentally, this ViewState property is the same as the one that was introduced in the chapter “Moving between Web Forms and Persisting State”. The StateBag class allows a series of key/value pairs to be entered, where the key is used to uniquely identify the item in the StateBag, and the value can be any variable.

For the SelectionTable control, the StateBag would need to store the items currently in the list, the color properties for selected and unselected items and the SelectedItem value. The “state” of the control must be stored to the ViewState StateBag just before, or just after the control is being rendered, so that the most current state is stored, and when the control is loaded, the state should be restored.

This needs to be done before any of the properties can be manipulated so that no new changes are overridden.

The following private subroutine will be used to save the state. Insert this into the SelectionTable class:

```
Private Sub SaveControlState()
  Me.ViewState.Item("Items") = Items
  Me.ViewState.Item("intSelectedItem") = intSelectedItem
  Me.ViewState.Item("colSelectedColor") = colSelectedColor
  Me.ViewState.Item("colUnselectedColor") = colUnselectedColor
End Sub
```

The code is fairly simple – it uses the StateBag's Item() property to manipulate the items in the bag. Using this method, if the item did not previously exist, it will be created, and if it existed previously, the item will be updated with the new value.

The subroutine to restore the state is equally simple:

```
Private Sub LoadControlState()
  If Not IsNothing(Me.ViewState.Item("Items")) Then
    Items = Me.ViewState.Item("Items")
  End If
  If Not IsNothing(Me.ViewState.Item("intSelectedItem")) Then
    intSelectedItem = Me.ViewState.Item("intSelectedItem")
  End If
  If Not IsNothing(Me.ViewState.Item("colSelectedColor")) Then
    colSelectedColor = Me.ViewState.Item("colSelectedColor")
  End If
  If Not IsNothing(Me.ViewState.Item("colUnselectedColor")) Then
    colUnselectedColor = Me.ViewState.Item("colUnselectedColor")
  End If
End Sub
```

Before each item is restored, an If condition checks to ensure that it does actually exist in the StateBag. It would probably suffice to only check for one item in the StateBag, as it could then be logically assumed that the other items were present, but to be safe, all four variables are checked before the assignments are made.

These two methods should be called from the CreateChildControls() and Page_Load() methods respectively. The CreateChildControls() method is as good a place as any for the SaveControlState() method to be called, as it is only executed once the control is to be rendered, so all property changes will have occurred by the time the method is called. The method should be called without any arguments before or after the code inside the CreateChildControls() method, as shown below:

```
Protected Overrides Sub CreateChildControls()
  '**Code to create table rows
  SaveControlState()
End Sub
```

Take note that the actual content creation code has been omitted and replaced by a comment.

The Page_Load event handler will appear with the LoadControlState() call as follows:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  LoadControlState()
End Sub
```

With these two methods and calls in place, the control can now manage its own state.

Using the State-Enabled Control

Add the control to a new web form, and set its SelectedColor and UnselectedColor properties using attributes:

```
<uc1:SelectionTable id="SelectionTable1" SelectedColor="LightBlue"
UnselectedColor="Gray" runat="server"></uc1:SelectionTable>
```

Add the appropriate declaration for the control in the class:

```
Protected WithEvents SelectionTable1 As SelectionTable
```

Now add two buttons to the form in the designer and name them btnPrevious and btnNext, and assign their Text properties to “<” and “>” respectively. Next add another button, name it btnRemove and set its Text property to “Remove”. Finally add a TextBox, txtNewItem, and another button, btnAdd, giving the button the Text, “Add Item”. Double click the btnPrevious button to hook up and event handler and insert the following code, so that the event handler looks like this:

```
Private Sub btnPrevious_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnPrevious.Click
  If SelectionTable1.SelectedItem <= 0 Then
    SelectionTable1.SelectedItem = SelectionTable1.Count - 1
  Else
    SelectionTable1.SelectedItem = SelectionTable1.SelectedItem - 1
  End If
End Sub
```

Add the following code for the btnNext button’s Click event handler:

```
Private Sub btnNext_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnNext.Click
  If SelectionTable1.SelectedItem >= SelectionTable1.Count - 1 Then
    SelectionTable1.SelectedItem = 0
  Else
    SelectionTable1.SelectedItem = SelectionTable1.SelectedItem + 1
  End If
End Sub
```

The btnRemove button’s Click event handler must look as follows:

```
Private Sub btnRemove_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRemove.Click
  If SelectionTable1.SelectedItem = SelectionTable1.Count - 1 Then
    SelectionTable1.RemoveItem(SelectionTable1.SelectedItem)
    SelectionTable1.SelectedItem = SelectionTable1.Count - 1
  Else
    SelectionTable1.RemoveItem(SelectionTable1.SelectedItem)
  End If
End Sub
```

The btnAdd button’s Click event handler should feature this code:

```
Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAdd.Click
  SelectionTable1.AddItem(txtNewItem.Text)
End Sub
```

And finally, the Page_Load event handler should initialize the SelectionTable:

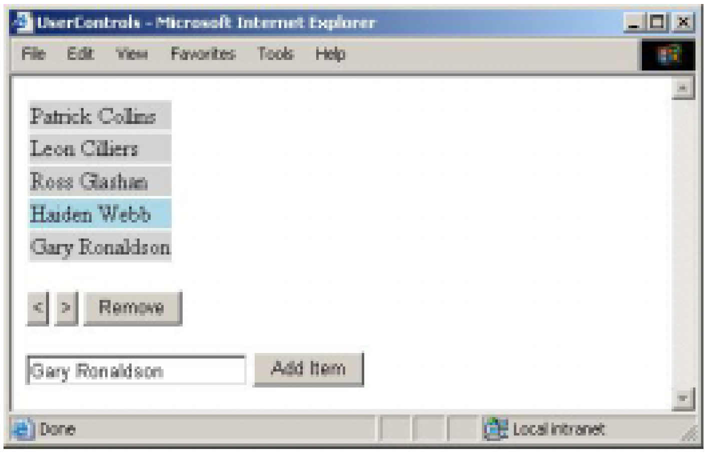
```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  If Not Page.IsPostBack Then
    SelectionTable1.AddItem("Patrick Collins")
    SelectionTable1.AddItem("Chris Hide")
    SelectionTable1.AddItem("Leon Cilliers")
    SelectionTable1.AddItem("Ross Glashan")
    SelectionTable1.SelectedItem = 0
  End If
End Sub
```

The basic purpose of this web form is to demonstrate how the page can interact with the user control across post-backs. The Previous and Next buttons allow the user to “cycle through” the different items in the list. The Remove button will remove the currently selected item from the list, and the Add button will add another item to the list using the text in the TextBox. The Previous and Next button event handlers both ensure that an item is always selected by checking to see if the currently selected item is at the start or end of the list and acting accordingly, For example, if the selected item is the last item of the list, and the user clicks the Next button, the event handler will select the first item, starting another “cycle”. The Previous button behaves similarly when the first item is selected and it is clicked.

The Remove event handler first checks to see if the currently selected item is the last on the list, so that when it is removed, the next item can be selected. This prevents a situation where no item is selected if the last item in the list is removed (except where it is the only item).

The Add button’s event handler simply call’s the control’s AddItem() method and passes the Text from the txtNewItem control. The Page_Load event handler first checks to see if the request is a postback, and if not (i.e. the page is being loaded for the first time), it populates the SelectionTable control with some values and selects the first item.

Figure 13.6 shows this demonstration form loaded in the browser after having been manipulated using the buttons available:



[Figure 13.6]

The use of the StateBag class and ViewState property is very important when maintaining state in control development, and can be a very useful tool for building robust, usable controls. Bare in mind that in order to achieve the state persistence for the control in the demo above, only several lines of code were required to save and load the state from the StateBag.

Creating Templated Controls

Templates are a completely new feature in ASP.NET that allow controls' user interfaces to be customised by XML "templates" in the page where the control is hosted. The Repeater control is based entirely upon templates, and many other controls, such as the DataList allow templates to be used to custom their appearance. The usefulness of this ability is particularly prevalent in these controls, but fortunately adding template functionality to custom controls isn't difficult.

To start off with, template functionality will first be added to the SelectionTable control to allow a custom "header" to be added to the table if the user wishes to do so. The first step to add template support is to expose a property of type ITemplate that will be used to store the template's content. The following code lists the private variable for storing the ITemplate value internally, and the actual property definition:

```
Private tmpHeaderTemplate As ITemplate = Nothing


<TemplateContainer(GetType(SelectionTable))> Public Property
HeaderTemplate() As ITemplate
  Get
    Return tmpHeaderTemplate
  End Get
  Set(ByVal Value As ITemplate)
    tmpHeaderTemplate = Value
  End Set
End Property
```

The only possible area that may cause confusion is the inclusion of an attribute. Attributes in VB.NET are denoted by the angled brackets (< and >), and are used to specify additional information about the structure. The TemplateContainer attribute is used to specify the type of the control that will contain the template.

Once the template property has been created, the CreateChildControls() method must be updated to insert the template if it was specified. The SelectionTable control will insert an extra row at the top of the table if the HeaderTemplate template is specified, so the code that should be added at the beginning of the CreateChildControls() method is as follows:

```
Protected Overrides Sub CreateChildControls()
  If Not IsNothing(tmpHeaderTemplate) Then
    Dim th As New HtmlTableRow()
    Dim tc As New HtmlTableCell()
    tmpHeaderTemplate.InstantiateIn(tc)
    th.Cells.Add(tc)
    tblSelectionTable.Rows.Add(th)
  End If
  ...
  ...
End Sub
```

The If condition checks to ensure that the HeaderTemplate property has been assigned a value. If it hasn’t then the tmpHeaderTemplate variable will still be “Nothing”. If a HeaderTemplate has been specified, then a new table row, th is created, along with a cell for the row, tc. The ITemplate interface’s InstantiateIn() method requires that a parameter of type Control is passed. The method then adds the contents of the template to this control. In the SelectionTable control, after the new table cell has been created and filled, it is added to the new row, and the row is added to the table. The remaining code from the original CreateChildControls() method follows.

Using the Templated Control

Using the new SelectionTable control with its header template functionality is very simple. Add the control to a new web form, and declare a protected member variable for it in the server-side code as per normal, to give the code programmatic access to the control. In the Page_Load event, initialise the control with some values:

```
Protected WithEvents SelectionTable1 As SelectionTable


Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  If Not Page.IsPostBack Then
    SelectionTable1.AddItem("The Rock")
    SelectionTable1.AddItem("Boiler Room")
    SelectionTable1.AddItem("The Matrix")
    SelectionTable1.AddItem("Double Jeopardy")
```

```
      SelectionTable1.SelectedColor = Color.LightGray
      SelectionTable1.SelectedItem = 2
    End If
  End Sub
```

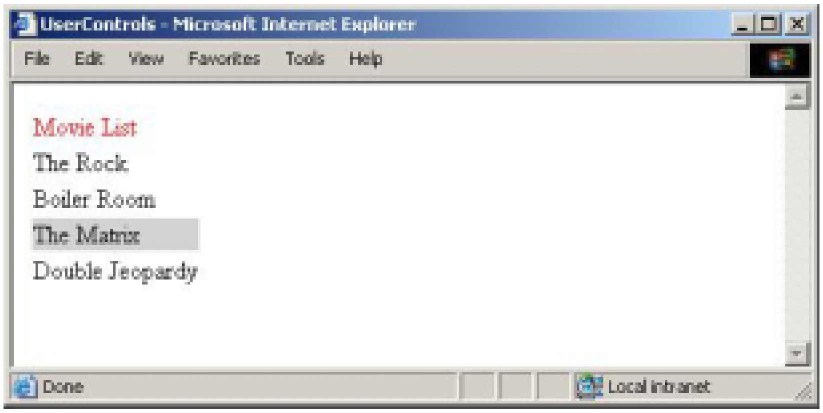
Switch back to the HTML view of the UI. The code for the control will look similar to this:

```
<uc1:SelectionTable id=SelectionTable1
runat="server"></uc1:SelectionTable>
```

Adding the HeaderTemplate to the control is fairly simple – all that needs to be done is a HeaderTemplate “tag” must be included as a “child” of the user control, and the code for the template must be specified inside:

```
<uc1:SelectionTable id=SelectionTable1 runat="server">
 <HeaderTemplate>
  <font color="#ff0000">Movie List</font>
 </HeaderTemplate>
</uc1:SelectionTable>
```

This will result in the text “Movie List” being inserted as a header for the table, in red text. However, the template could contain practically any HTML or even ASP.NET server controls. Figure 13.7 shows the web form with the templated SelectionTable control loaded:



Movie List
The Rock
Boiler Room
The Matrix
Double Jeopardy

[Figure 13.7]

Inheriting From and Extending Existing Controls

One of the marvels of object-oriented programming is the ability for one class to derive from another and inherit the base class’ members and functionality, and then extend upon it. In fact, all the controls in this chapter thus far have been derived from the System.Web.UI.UserControl class. However, it may be useful to derive from an even higher level class, such as the TextBox or HtmlContainerControl. Using such methods, it is possible to create a control quickly and easily, without even needing to use .ascx files.

An “HtmlMenuItem” Control

Many websites build their menus from databases, so the menu items have to be generated programmatically – some from the database, and the others are hard coded. Assume that a menu with the following structure is desired, where each item is a link:

```
Home | Members | Articles | News | Events | Jobs | About
```

Instead of simply generating an HTML string to add to the page to create the menu, it would be possible to build an HtmlMenuItem control, derived from the System.Web.UI.HtmlControls.HtmlContainerControl class, which would offer an object-oriented approach to building the menu. The class in Listing 13.5 implements the HtmlMenuItem object, and shows both how to derive from a regular Html Control class, and how to create a new control entirely programmatically:

```
Public Class HtmlMenuItem
  Inherits System.Web.UI.HtmlControls.HtmlContainerControl

  Private strItemName As String = ""
  Private strLinkURL As String = ""
  Private boolVbar As Boolean = True

  Public Property ItemName() As String
    Get
      Return strItemName
    End Get
    Set(ByVal Value As String)
      strItemName = Value
    End Set
  End Property

  Public Property LinkURL() As String
    Get
      Return strLinkURL
    End Get
    Set(ByVal Value As String)
      strLinkURL = Value
    End Set
  End Property

  Public Property Vbar() As Boolean
    Get
      Return boolVbar
    End Get
    Set(ByVal Value As Boolean)
      boolVbar = Value
    End Set
  End Property

  Public Sub New(ByVal MenuItemName As String, ByVal
MenuItemLinkURL As String)
    ItemName = MenuItemName
    LinkURL = MenuItemLinkURL
  End Sub

  Public Sub New(ByVal MenuItemName As String, ByVal
MenuItemLinkURL As String, ByVal MenuItemVbar As Boolean)
    ItemName = MenuItemName
    LinkURL = MenuItemLinkURL
    Vbar = MenuItemVbar
  End Sub

  Protected Overrides Sub CreateChildControls()
    Dim anchor As New HtmlAnchor()
    anchor.InnerText = ItemName
    anchor.HRef = LinkURL
    Me.Controls.Add(anchor)
    If Vbar Then
      Me.Controls.Add(New LiteralControl("&nbsp;|&nbsp;"))
    End If
  End Sub

End Class
```

[Listing 13.5]

The “Inherits” keyword is used to derive the control from the HtmlContainerControl class. The next few lines declare the private variables for storing the three property values, and the properties themselves – ItemName, LinkURL and Vbar. ItemName contains the name that will be displayed, LinkURL contains the URL that the menu item must link to, and the Vbar property is a boolean specifying whether or not a vertical bar should be appended to the end of the item (the last item in the menu should not have one, which is why this is needed).

Two constructors are available. Both simply take the arguments passed to them and assign them to the appropriate properties. Finally, the overridden CreateChildControls() method creates the actual HTML for the item just prior to it being rendered. An HtmlAnchor control is used to create the link, which is added to the control’s Controls list, and if necessary, a vertical bar is included, preceded and followed by non-breaking spaces.

To actually use the new control, create a new class in Visual Studio.NET by right-clicking the project node in the Solution Explorer, and click Add-Add New Item and choose “Class” from the dialog. Finally, insert the implementation from Listing 13.5 into the class. Other pages in the project can then use the class.

To try out the HtmlMenuItem class, create a new web form, and add a label control named, “lblMenu”. Ensure that the Text property is empty. Switch to code view, and insert the following code into the Page_Load event:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  lblMenu.Controls.Add(New HtmlMenuItem("Home", "default.aspx"))
  lblMenu.Controls.Add(New HtmlMenuItem("Members", "members.aspx"))
  lblMenu.Controls.Add(New HtmlMenuItem("Articles", "articles.aspx"))
  lblMenu.Controls.Add(New HtmlMenuItem("News", "news.aspx"))
  lblMenu.Controls.Add(New HtmlMenuItem("Events", "events.aspx"))
  lblMenu.Controls.Add(New HtmlMenuItem("Jobs", "jobs.aspx"))
  lblMenu.Controls.Add(New HtmlMenuItem("About", "about.aspx", False))
End Sub
```

When the page is loaded, the menu will be generated and displayed. Of course for this example, the control is fairly pointless, as the menu could easily have been generated in the form designer using the regular controls, however, it’s usefulness becomes more evident when the menu, or parts thereof, must be loaded from a database, and even more so when items in the menu might be subject to change their display name or URL, thanks to the inclusion of properties in the control.

Deriving from and extending existing controls can be extremely powerful, and save a significant amount of time. For example, the AdRotator control is a fine base to work from when attempting to build a custom banner control. It already contains much of the logic for a simple banner rotation system, and could possibly be modified for more individual needs, such as loading the banner data from a SQL Server database rather than an XML file without nearly as much effort as it would take to re-implement the control from scratch.

Conclusion

This chapter has provided a detailed overview of much of what is possible using user controls in ASP.NET. Custom-built user controls are an essential part of code-reuse in the ASP.NET programming model, and offer a solution that is significantly more comprehensive and powerful than the existing ASP equivalent of server-side includes. Controls should not be over-used, but for situations where code reuse is bound to occur, such as with basic page layout items like headers, footers and menus, controls make an obvious solution to the inevitable maintenance problems if a reusable design element is not used. The ability to persist state across postbacks, use the new templates functionality and inherit from other controls all add to the appeal of user controls as a medium for encapsulation and code reuse.

Chapter 14

Web Services

Much of the hype following the dot-com fallout has been related to B2B (Business-to-Business) Internet transactions, and in particular "Web Services". This chapter aims to remove the marketing hype from the topic and present a clear case of exactly what web services are, how they work, how they help you develop better applications and how you can build (and consume) them using Microsoft's technologies.

What are “Web Services”?

Up until the arrival of web services, the Internet has served primarily as a method for businesses communicating with customers. These communications range from simple "brochure"-type websites that are static and simply offer information, to fully interactive, database-driven websites that allow the viewer to interact with the site and gain specific information, or perform certain tasks. Examples of such sites include web-based email, web-based personal organisers and e-commerce sites. An important part of running a website is that people expect the information on the website to be timely and accurate, amongst many other things. Maintaining a website often consumes large amounts of time to ensure that these two expectations are met. However, matters are only made worse for websites that rely on information from 3rd parties. An example of this would be that an e-commerce requires up-to-date shipping prices from its courier service to that is can deliver accurate shipping prices to customers using the site. Currently what most e-commerce sites do is store the courier price matrix on their website (in a database, or in plain logic code). What happens when the courier changes their prices? The website's developers will have to modify their version of the courier's prices. This unnecessary duplication fomrs a large part of the maintenance time of websites that deal with 3rd party information.

A Different Approach

The Object Oriented Programming paradigm was a huge advance in the world of computer programming. It advocated the principle of code-reuse. "Why rewrite a piece of code, when you've already written it elsewhere?". The principles of OOP were advanced upon and incorporated into an era of "component oriented programming" where specific reusable functionality is contained in components. The .NET framework and VS.NET fully embrace this programming paragdim, and C# was even developed specifically with this in mind. Much of the thinking behind web services revolves around the same principles, which are again, expanded upon. The theory of web services is that web sites should not only provide information to customers, but also expose certain business logic and information to business partners. Thinking back to the hypothetical e-commerce sites that stores its courier's rates itself, an application of a web service could be that the courier company exposes its shipping rates as a web service, that the e-commerce site could use to obtain the most recent rates on-the-fly whenever a quote for a customer is generated. Figure 14.1 shows the flow of information in this scenario.

[Figure 14.1 - An example of a simple web service]

However, using web services to transfer information between websites is only scratching the surface of what web services are capable of. They could be used in a more active role, such as performing B2B e-commerce transactions. To get an idea of what is possible, consider a hypothetical book retailer called ABC Books. This company relies of XYZ Freight to do its deliveries to customers. Now imagine a customer placing an order for a book on the ABC Books website in a "web services world". Firstly, when the customer goes to the ordering section and enters their shipping details, the site will use a web service exposed by the XYZ Freight website to obtain the shipping details for the client. The shipping rate is guaranteed to be accurate, as it is coming straight from the freight company. The customer then decides to order the book. In addition to the regular credit card payment processing, the site will also use another web service exposed by the XYZ Freight website that places a shipping order with the freight company, detailing the mass, original location and destination. The XYZ Freight web service could then theoretically use web services exposed by the freight airline companies that it uses to make sure that this latest order is included in its next shipment from and to the required destinations. Figure 14.2 outlines the entire process.

[Figure 14.2 - Hypothetical use of web services for an online book retailer]

As can be seen, web services are not only about code and information reuse *across companies and websites* but also about using the web to automate business processes. Naturally the extent to which this can be done depends entirely on the business under discussion, but to a lesser or greater extent all businesses that reply on 3rd party services, where both parties have "web service enabled" websites, can greatly benefit from this new technology. This is an area where the marketing hype can be confusing and seems to imply that web services are going to totally change every aspect of business on the internet. This is untrue, and although web services are a great idea and will most likely help many businesses, they're not the final solution to all things Internet-related. Just as OOP and component oriented programming do not alleviate the need to write code but do make programmers' lives easier, so to do web services open a new range of possibilities and make programming B2B applications easier, they're not the solution to all web programming woes, as some marketing campaigns would have you believe.

Chapter 15

Caching

As anyone who uses the Internet will tell you, performance is a major issue. As a developer, everything must be done to optimize application performance as much as possible, to reduce the strain on server resources so that more clients can be served. In single user, low-load applications, there is no need for caching. However, as soon as external resources are accessed, caching becomes essential, particularly in n-tiered web application development. Previously most caching functionality had to be programmed manually, but ASP.NET now provides two extensive caching mechanisms which should provide enough functionality to cover any caching requirement without any additional “plumbing” code. These two types of caching are Output Caching and Data Caching. Output Caching is the more basic of the two, and uses a directive to turn caching on or off for a particular page. This method is ideal for entire pages that need to be cached for certain periods of time. Data Caching has a more specific application. It is used for storing specific pieces of data, which can include objects, and it programmatically accessible. As a vast oversimplification, it can be considered as a special type of Application variable.

Output Caching

Output Caching provides an extremely simple method of caching entire pages, or in fact, even parts of pages. Output Caching is enabled by using the @ OutputCache directive in the pages that need to be cached. The syntax of this directive is as follows:

```
<%@ OutputCache Duration="#ofseconds" Location="Any | Client |
Downstream | Server | None" VaryByControl="propertyname"
VaryByCustom="browser | customstring" VaryByHeader="headers"
VaryByParam="parametername" %>
```

As can be seen, the directive offers numerous parameters for modifying its functionality. These parameters make this facility very flexible and suitable for use in most page and control caching scenarios without any additional programming.

Output Caching Parameters

Duration

The most basic use of this OutputCache directive involves the Duration parameter and VaryByParam parameter. These parameters are mandatory when using output caching. Duration specifies the length of time (in seconds) for which the page will be cached before it is refreshed. VaryByParam indicates for which querystring parameters the page must be refreshed. Again, even if you don’t intend to differ the cached versions for different values passed through querystrings, this parameter is mandatory and omitting it will cause a compiler error. Listing 16.1 demonstrates the simplest form of Output Caching:

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="OutputCachingDemo.aspx.vb"
Inherits="Caching.OutputCachingDemo"%>
<%@ OutputCache Duration="60" VaryByParam="None" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
    <HEAD>
        <title></title>
        <meta name="GENERATOR" content="Microsoft Visual
Studio.NET 7.0">
        <meta name="CODE_LANGUAGE" content="Visual Basic
7.0">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
    </HEAD>
    <body>
        <form id="frmOutputCachingDemo" method="post"
runat="server">
            <!-- Data from dynamic source, such as a database -->
        </form>
    </body>
</HTML>
```

This page doesn’t actually contain anything that would make caching advantageous, such as data pulled from a database, or information from a web service, but only demonstrates the use of the directive. As can be seen, caching is enabled in one line. No extra programming is required. It has a Duration of 60 seconds, meaning that 60 seconds after the first time this page is hit, it will be refreshed. All hits during those 60 seconds will be served from the cached page. Had information from a database been included in the page, the database server would not have been hit, as the entire page would’ve been loaded from the cache, including all the dynamic content. An easy way to test that the page is in fact being cached, and subsequently loaded from the cache is to put a timestamp on the page, which will show when the page was last generated. You can do this by adding a Label Web Control to the form with the name lblLastGenerated and then add this code to the Page_Load event in the Code Behind file:

```
lblLastGenerated.Text = DateTime.Now.ToString(“G”)
```

When the page is loaded, the label’s text property is assigned the current date and time in the format “mm/dd/yyyy hh:mm:ss”. The first time the page is loaded, the label will display the current date and time. However, subsequent hits in the next 60 seconds will display the same date and time as the original hit because the page is being loaded from the cache, rather than reprocessing the page. After the 60 second time interval has elapsed, the next hit will cause the page to again to regenerated and the current date and time will be displayed, and so forth.

VaryByParam

Even simple Web Forms would often not be able to take advantage of output caching if the VaryByParam parameter was not available. Very often the contents of a page will change according the various querystring parameters being passed to it. This is not to say that the contents change for subsequent hits to the page with the same querystring parameters however, which is why caching may be appropriate. The VaryByParam parameter for output caching is available to cater for almost all querystring-related scenarios. Listing 16.2 shows an ASP.NET page that uses the VaryByParam parameter, along with Listing 16.3 – the codebehind for 16.2 – to make effective use of caching whilst ensuring that accurate data is delivered to the user.

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="OutputCachingDemo.aspx.vb"
Inherits="Caching.OutputCachingDemo"%>
<%@ OutputCache Duration="60" VaryByParam="catid" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
    <HEAD>
        <title></title>
        <meta name="GENERATOR" content="Microsoft Visual
Studio.NET 7.0">
        <meta name="CODE_LANGUAGE" content="Visual Basic
7.0">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
    </HEAD>
    <body>
        <form id="frmOutputCachingDemo" method="post"
runat="server">
            <asp:Panel id="pnlProducts"
runat="server"></asp:Panel>
        </form>
    </body>
</HTML>
```

[Listing 16.2]

```
Public Class OutputCachingDemo
  Inherits System.Web.UI.Page
  Protected WithEvents pnlProducts As System.Web.UI.WebControls.Panel


#Region " Web Form Designer Generated Code "


  'This call is required by the Web Form Designer.
```

```
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

    End Sub

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Dim objConn As New
System.Data.OleDb.OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0
;Data Source=c:\Northwind.mdb")
        Dim objCommand As New
System.Data.OleDb.OleDbCommand("SELECT ProductID, ProductName
FROM Products WHERE CategoryID = " & Request.QueryString("catid"),
objConn)
        objConn.Open()
        Dim objDR As System.Data.OleDb.OleDbDataReader =
objCommand.ExecuteReader()
        While objDR.Read()
            Dim objLink As New System.Web.UI.WebControls.HyperLink()
            objLink.NavigateUrl = "viewproduct.aspx?productid=" &
objDR.GetValue(0)
            objLink.Text = objDR.GetString(1)
            pnlProducts.Controls.Add(objLink)
            pnlProducts.Controls.Add(New System.Web.UI.LiteralControl("<br
/>"))
        End While
        objDR.Close()
        objConn.Close()
    End Sub

End Class
```

[Listing 16.3]

The ASP.NET page includes an OutputCache directive like this:

```
<%@ OutputCache Duration="60" VaryByParam="catid" %>
```

This tells the compiler to cache the page output for 60 seconds for requests where the querystring parameter “catid” is the same. In other words, if the page is called by a client using OutputCachingDemo.aspx?catid=1 then the output from that page is cached. Subsequent requests for OutputCachingDemo.aspx?catid=1 for the following 60 seconds are loaded from the cache. However, calls to the page with other values for catid will not be loaded from the cache. For example, a call to OutputCachingDemo.aspx?catid=2 would regenerate the page and store the result when catid is 2 in cache. Subsequent requests to OutputCachingDemo.aspx?catid=2 in the next 60 seconds will be loaded from cache.

VaryByParam for multiple parameters

The VaryByParam parameter for Output Caching provides for a great amount of flexibility, but there may be instances where more than one querystring parameter has effect one what should be displayed on the page. VaryByParam provides for this eventuality in one of two ways – either the required parameters can all be specified, separated by semi-colons, or an asterix can be used, effectively specifying all the querystring parameters. An example of the usage for the former would be as follows:

```
<%@ OutputCache Duration="60" VaryByParam="catid,supplierid" %>
```

In this case, any change to either of the catid or supplierid querystring parameters will cause the page to be generated and not loaded from cache. For example, a call to OutputCachingDemo.aspx?catid=1&supplierid=1, then OutputCachingDemo.aspx?catid=1&supplierid=2 will cause two different versions of the page to be cached. In this case, both the catid and supplierid querystring parameters must have the same values if the page is to be loaded from cache. This would not be possible if only one querystring parameter was specified, for example as follows:

```
<%@ OutputCache Duration="60" VaryByParam="catid" %>
```

Because changes to the supplierid querystring would be ignored. In this instance calls to OutputCachingDemo.aspx?catid=1&supplierid=1 and OutputCachingDemo.aspx?catid=1&supplierid=2 would render the same page if loaded from cache, which is obviously not the desired result.

The following example demonstrates the use of the asterix to specify “all querystring parameters”:

```
<%@ OutputCache Duration="60" VaryByParam="*" %>
```

Calls to the page where any querystring parameter’s value is different will result in a new version of the page being cached. Requests must be identical (in terms of the values passed through the querystring, at least) in order for the page to be loaded from cache. This can be very useful in some circumstances, but preferably only use it where it makes sense to, as pages that take in many querystring parameters may strain the server.

VaryByHeader

Every request to a file on a web server sends with it specific information. This includes information about the browser that is requesting the page, the localization settings on the client and other information. Table 16.1 shows the list of so-called common HTTP “header” variables.

HTTP Header	Description
Accept	The types of files that the client will accept
Accept-Charset	If the client can accept special-purpose character sets, this header will be used to indicate the capability to the server.
Accept-Encoding	The content-coding values that the client will accept
Accept-Language	The language setting that the client would prefer the result to be returned in (this is normally the language setting of the client machine)
Connection	The type of TCP/IP connection
Host	The name/IP address of the server
Referer	The URI of the page that caused this request to be made.
User-Agent	Information about the client browser

[Table 16.1 – Common HTTP header variables]

The VaryByHeader parameter can be very useful in certain circumstances, particularly when used in applications that are localized. The VaryByHeader parameter works very similarly to the VaryByParam parameter, in that it allows for different versions of a page to be cached, depending on how the page was requested. The VaryByHeader parameter will not be used very often, although it can play a very important role in localized applications.

> A localized application is an application that tailors its output to clients according to their localization settings, which is normally the native language setting, or dialect thereof, such as en-ZA, which is South African English. In .NET, this is known as a ‘culture’ and not only allows for language customization, but also calendars, daylight savings time and other culture-related information.

As has already been mentioned, all OutputCache directives must include the VaryByParam parameter, even it its functionality is not required. In the example in listings 16.4 and 16.5 (aspx and VB codebehind files respectively), the use of only the VaryByHeader parameter is demonstrated for localization purposes, but the VaryByParam parameter is notably present.

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="OutputCachingDemo.aspx.vb"
Inherits="Caching.OutputCachingDemo"%>
<%@ OutputCache Duration="60" VaryByParam="None"
VaryByHeader="Accept-Language" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
        <HEAD>
                <title></title>
                <meta name="GENERATOR" content="Microsoft Visual
Studio.NET 7.0">
                <meta name="CODE_LANGUAGE" content="Visual Basic
7.0">
                <meta name="vs_defaultClientScript" content="JavaScript">
                <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
        </HEAD>
        <body>
                <form id="frmOutputCachingDemo" method="post"
runat="server">
                        <asp:Label id="lblEnglishName"
runat="server"></asp:Label>
                </form>
        </body>
</HTML>
```

[Listing 16.4]

```
Public Class OutputCachingDemo
  Inherits System.Web.UI.Page
  Protected WithEvents lblEnglishName As
System.Web.UI.WebControls.Label

#Region " Web Form Designer Generated Code "

  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

  End Sub

  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub

#End Region

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
```

```
    Dim objCultureInfo As New
System.Globalization.CultureInfo(Request.ServerVariables("HTTP_ACCEP
T_LANGUAGE"))
    lblEnglishName.Text = objCultureInfo.EnglishName
  End Sub

End Class
```

[Listing 16.5]

This example varies the cache according to the Accept-Language HTTP header. In essence this means that versions of this form will be cached for requests from different parts of the world. This isn’t entirely correct, as client systems can “lie” about where the come from, but for the purpose of this example, this is what happens. This page doesn’t actually do very much with the localization information that it receives and doesn’t load the page content from a database accordingly, or anything similar, but simply demonstrates how a page could be cached according to the origin of the request. Let’s assume that the page received hits with the Accept-Language header with the following values, in this sequence:

en-US

en-GB

en-GB

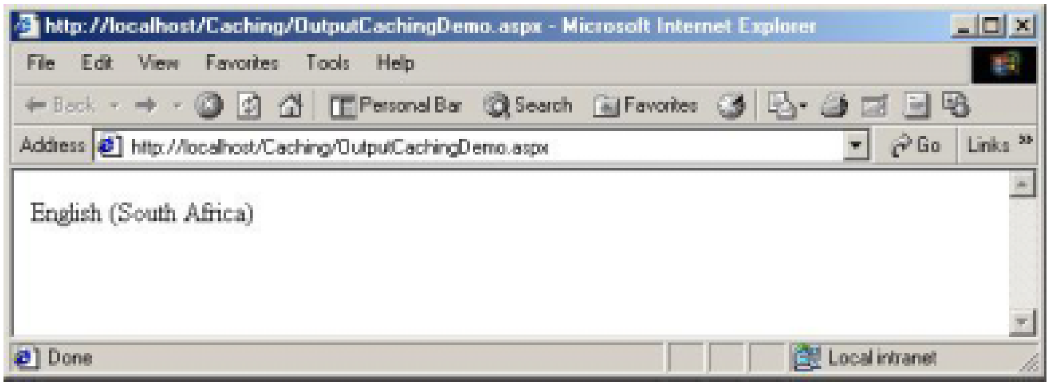
en-ZA

en-US

en-GB

en-US

Assuming that the hits were within a 1-minute time period, 3 versions of the page’s output would be cached – one for en-US, one for en-GB and one for en-ZA, with the remaining hits all be served from the cache. To test this example for different localization settings, change the Duration to 600 (10 minutes) so that you will have enough time to change your localization settings without your cache being invalidated. To make sure that you are receiving cached pages when you’re supposed to be, you make wish to add a label Web Control to the form and set it’s ‘Text’ property to the current time when the page it loaded. Load the page with your current localization settings. i.e. Don’t change anything yet – just load the page. A version for your localization setting will be cached. The full English name for your localization setting should be displayed, as shown in Figure 16.1.

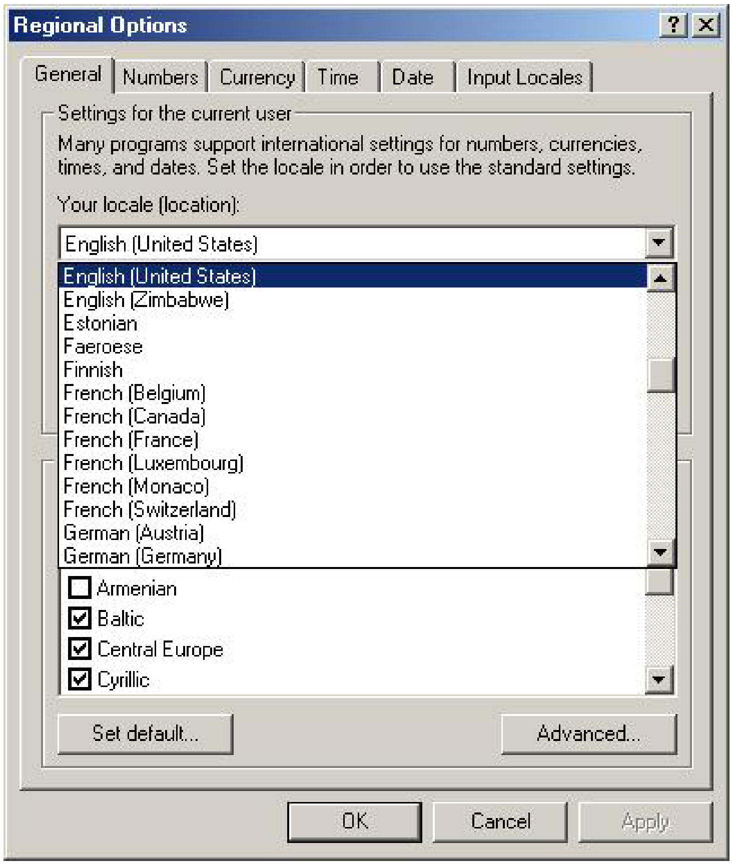

http://localhost/Caching/OutputCachingDemo.aspx - Microsoft Internet Explorer

Address: http://localhost/Caching/OutputCachingDemo.aspx

English (South Africa)

[Figure 16.1 – Output Caching with localization settings]

In order to change your localization settings, go to your Control Panel and open the Regional Options dialog. It should look similar to Figure 16.2:


Regional Options

Settings for the current user

Many programs support international settings for numbers, currencies, times, and dates. Set the locale in order to use the standard settings.

Your locale (location): English (United States)

English (United States)
English (Zimbabwe)
Estonian
Faeroese
Finnish
French (Belgium)
French (Canada)
French (France)
French (Luxembourg)
French (Monaco)
French (Switzerland)
German (Austria)
German (Germany)

☐ Armenian
☑ Baltic
☑ Central Europe
☑ Cyrillic

[Figure 16.2 – Regional Options in Windows 2000]

It is here that you can change your locale, which IE sends as the Accept-Language parameter in all HTTP requests. Change the setting to something other than your original locale and click Ok. Now refresh the page in IE – the locale that you chose should be displayed. However, if you change the locale back to its original setting and refresh the page, the page will now be loaded from cache.

This example is not very practical, as it doesn't actually use the localization data, but in a real-world situation, the data could be used to obtain the contents of the page from a database, or something similar, where caching would be useful when used in conjunction with the values of HTTP headers to increase the performance of the application.

Location

The HTTP 1.1 protocol provides built-in functionality for caching, in the form of HTTP headers. The “Location” attribute of the OutputCache directive allows for flexibility in specifying where cached copies of the page will be stored, which may involve the use of HTTP headers. The headers concerned are the “Cache-Control” and “Expires” headers. The “Cache-Control” header, when used in combination with the “Expires” header can be used to specify if and where the page in question can be cached. This includes whether the page cannot be cached at all, whether it can be cached on only the client or on the client and proxy servers between the client and the server. ASP.NET alleviates the necessity to deal with the headers at all by providing the “Location” attribute, which, in addition to using HTTP headers to define caching policy, also makes use of the ASP.NET engine. To understand the “Location” attribute more clearly, and the respective values that it can contain, it is important to understand that when a client requests a page, the request, and thus response, more that likely goes through a series of machines, and the request does not go directly to the server (this may be the case in some Intranet scenarios, but on the Internet, this is highly unlikely to occur). Figure 16.3 shows a highly simplified diagram of a request and response for a page to demonstrate this concept.

[Figure 16.3 – Following the page of an example HTTP request/response on the Internet]

As can be seen, Internet traffic from the client will typically go through numerous “proxy” servers at the user’s ISP, and possibly even of the company that are hosting the actual website. One of the functions of proxy servers is to cache the content that goes through them such that subsequent requests for previously requested content are delivered from the proxy server, rather than re-fetching the content from the web server. However, before HTTP 1.1-compliant proxy servers cache content, they ensure that the HTTP headers specify that they are allowed to do so (or more specifically, they check to see if they are not allowed to, and if they are not, then they do not cache the content, but all other content is cached).

The “Location” attribute has 4 possible values. These values stored in the System.Web.UI.OutputCacheLocation enumeration. Table 16.2 shows the possible values and their purpose.

Value	Purpose
Any	The cache can be located on the client, a proxy server or on the web server
Client	The cache is located on the browser
Downstream	The cache is located on proxy servers between the client and the server
None	The page may not be cached
Server	The cache is located on the web server

[Table 16.2 – OutputCacheLocation enumeration values]

The default value (the value if the “Location” attribute is omitted) is “Any”. In most cases, it is unnecessary to modify this property. The most common instance where this is not the case is in applications where a user may be referred back to a page whose content is likely to have changed since their last hit. In this case, it is undesirable for the browser to cache the page (in which case the user has to perform a manual “refresh” to see the updated page), and therefore should be instructed to load the file from the server for all requests (be it generated, or from the server cache) and not the browser cache. In a scenario such as this, and OutputCache directive might be used as follows:

```
<%@ OutputCache Duration=”60” VaryByParam=”stockquotesymbol”
Location=”Server” %>
```

In an application such as a stock quote graph generator, it is important that if the user is redirected to the generation page, they receive a page that is sent from the server, and not loaded from their browser’s cache. Using the “Server” value for the “Location” attribute achieves this result.

Another use of the “Location” attribute is to disable caching entirely for the page. By default pages may be cached on downstream (proxy) servers and on the client. If there is a scenario where there should be no caching, then the “None” value should be used, and will disable caching on the server, as well as downstream servers and the client. An example of this usage would be:

```
<%@ OutputCache Duration=”60” VaryByParam=”None” Location=”None”
%>
```

There are other occasions where this functionality can prove useful. However, it is not often that it is required, and can normally be safely left to use the default value of “Any”, but it is useful to know in case of the two abovementioned scenarios, or other similar instances where you need control over where cached copies of pages are stored.

VaryByCustom

Until now, the Output Caching mechanism has provided numerous properties to make it more flexible, but the “VaryByCustom” attribute is the definitive solution to most possible shortcomings of the Output Caching feature in ASP.NET. The purpose of VaryByCustom is actually twofold – firstly, it provides a mechanism to allow different versions of pages to be cached according to the client browser that requests them, and secondly, to allow you, the developer, to specify any criteria for caching that has not already been covered by the other attributes.

“VaryByBrowser”

A major potential problem with output caching is that a page could be cached after being requested by a DHTML-aware browser, such as IE5, and thus contain a large amount of client-side code (such as validation code), and then be requested by a downlevel browser such as IE3, or Netscape 3, which does not support DHTML, and thus return numerous scripting errors on the client-side, or simply not run at all when the server returns the cached version of the page. To alleviate this problem, Microsoft has included the VaryByCustom attribute with the value of “browser”, which will cache different versions of a page for each browser, using the browser name and major version as its criteria. This means that a page requested by IE5.0, IE5.01 and Netscape 4 would have 2 versions cached – one for IE5, one for Netscape 4. In other words, VaryByCustom=”browser” only uses the major browser version as a factor, and not minor browser versions.

As has already been mentioned, validation is one area where caching could cause client scripting errors. However, the effects of caching should be taken into account in any application where client-side scripting is generated on-the-fly by ASP.NET code according to the browser version. This is a common technique, also often used to dynamically deploy stylesheets and other browser-version specific elements. VaryByCustom with the “browser” value enables pages that use these techniques to take advantage of the powerful caching features of ASP.NET without interfering with the existing browser-specific code.

Listing 16.6 and 16.7 list the code demonstrating a simple use of VaryByCustom=”browser”, where the name of the client browser and last time generated are displayed and cached for 60 seconds for each different browser that hits the page.

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="OutputCachingDemo.aspx.vb"
Inherits="Caching.OutputCachingDemo"%>
<%@ OutputCache Duration="60" VaryByParam="None"
VaryByCustom="browser" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
    <HEAD>
        <title></title>
        <meta name="GENERATOR" content="Microsoft Visual
Studio.NET 7.0">
        <meta name="CODE_LANGUAGE" content="Visual Basic
7.0">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
    </HEAD>
    <body>
        <form id="frmOutputCachingDemo" method="post"
runat="server">
            <P>
                <asp:Label id="lblBrowser"
runat="server"></asp:Label>
            </P>
            <P>
                <asp:Label id="lblLastGenerated"
runat="server"></asp:Label>
```

```
                        </P>
                </form>
        </body>
</HTML>
```

[Listing 16.6]

```
Public Class OutputCachingDemo
  Inherits System.Web.UI.Page
  Protected WithEvents lblLastGenerated As
System.Web.UI.WebControls.Label
  Protected WithEvents lblBrowser As System.Web.UI.WebControls.Label

#Region " Web Form Designer Generated Code "

  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

  End Sub

  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub

#End Region

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    lblBrowser.Text = Request.Browser.Browser
    lblLastGenerated.Text = DateTime.Now.ToString("T")
  End Sub

End Class
```

[Listing 16.7]

Assuming that the hits occurred in the following order, listed by the client browser used:

IE 5

IE 5

Netscape 4

Netscape 3

IE 6

Netscape 4

4 versions of the page would be cached – one for IE 5, one for IE 6, one for Netscape 3 and one for Netscape 4. Since the caching duration is set at 60 seconds, the hits for each respective browser would have to be within the 60-second limit before a new version would be generated for the requesting browser type and version.

Custom Caching

The VaryByCustom attribute offers, as the previous section details, a method of generating different versions of a page according to the requesting browser. However, the “browser” value for the VaryByCustom attribute is the only intrinsic value available. However, the true power and value of VaryByCustom is derived from its namesake – it allows you, the developer, to define custom criteria that the caching mechanism will use to determine whether to regenerate a page or to load it from the cache. This offers a huge amount of flexibility and allows for any situation not already covered by the previous attributes to be handled. Flexibility does however come at a price, and in this case, the price is that implementing VaryByCustom with custom dependencies is that it is not quite as simple as previous implementations where only one line in the page was required. VaryByCustom requires a visit to the application’s Global.asax file to define exactly what the custom criteria are.

The Global.asax file was covered in detail in the previous chapter, where it mentioned that the Global.asax file was used, amongst other things, to override methods of the HttpApplication class that it is inherited from to customise the behaviour of the application. GetVaryByCustomString is a method exposed by the HttpApplication class and is used by ASP.NET to obtain a string that was generated using the custom caching criteria. This string is then returned, just as the contents of a required querystring variable in the case of VaryByParam would be, and compared against those of the currently cached copies of the page to see whether a cached copy should be served, or a new version should be generated. GetVaryByCustomString takes two parameters – an instance of an HttpContext class and a string. The HttpContext class contains information about the request at hand, all in one convenient object, and the “arg” string parameter contains the argument that was passed in the ASP.NET page’s VaryByCustom argument. For example, for the following line in an ASP.NET, the “arg” value passed would be “frames”:

```
<%@ OutputCache Duration="60" VaryByParam="None"
VaryByCustom="frames" %>
```

All of the calling of GetVaryByCustomString and the passing of arguments etc is done behind the scenes by ASP.NET. The only part of the process that requires attention is the actual GetVaryByCustomString method itself. It needs to be overridden and return a value according to the “arg” parameter that is passed into it. For example, let’s assume that our VaryByCustom value is “frames”. If the requesting browser supports frames, then a page will be loaded from cache that uses frames, or such a page will be generated, and if the requesting browser does not support frames then the respective page will be loaded from cache or generated. The GetVaryByCustomString method will be passed “frames” as the “arg” parameter, and the method must return “frames=true” or “frames=false”. Because an HttpContext object is also passed, it is possible to quickly and easily obtain the frames capability of the requesting browser through the HttpRequest’s Browser Frames property that is a property of the HttpContext class. Listing 16.8 shows a Global.asax codebehind file with the GetVaryByCustomString method overridden providing functionality for a VaryByCustom attribute value of “frames”.

```
Imports System.Web
Imports System.Web.SessionState

Public Class Global
  Inherits System.Web.HttpApplication

#Region " Component Designer Generated Code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Component Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

  End Sub

  'Required by the Component Designer
  Private components As System.ComponentModel.Container

  'NOTE: The following procedure is required by the Component Designer
  'It can be modified using the Component Designer.
  'Do not modify it using the code editor.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    components = New System.ComponentModel.Container()
  End Sub

#End Region

  Sub Application_BeginRequest(ByVal sender As Object, ByVal e As
EventArgs)
    ' Fires at the beginning of each request
  End Sub
  Sub Application_AuthenticateRequest(ByVal sender As Object, ByVal e As
EventArgs)
    ' Fires upon attempting to authenticate the use
  End Sub

  Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires when an error occurs
  End Sub

  Public Overrides Function GetVaryByCustomString(ByVal Context As
HttpContext, ByVal arg As String) As String
    Return "frames=" & Context.Request.Browser.Frames
  End Function

End Class
```

[Listing 16.8]

An ASP.NET page in the application could then include the following OutputCache directive, and one version of the page would be cached if the requesting client browser supported frames, and another version if a request was made by a browser that did not.

```
<%@ OutputCache Duration="60" VaryByParam="None"
VaryByCustom="frames" %>
```

Like all the other “Vary” attributes, VaryByCustom supports multiple values, separated by commas. This enables more that one criterion to be “required” for a cached copy to be retrieved. Continuing with the example of caching based on browser functionality, we’ll add a new criterion to the list in addition to “frames” – “applets”, which will create different versions of pages in the cache according to whether the client browser supports java applets or not. Listing 16.9 shows the updated GetVaryByCustomString to allow for this extension.

```
Public Overrides Function GetVaryByCustomString(ByVal Context As
HttpContext, ByVal arg As String) As String
  If arg = "frames" Then
    Return "frames=" & Context.Request.Browser.Frames
  ElseIf arg = "applets" Then
    Return "applets=" & Context.Request.Browser.JavaApplets
  Else
    Return ""
  End If
End Function
```

[Listing 16.9]

If the “arg” parameter passed to the function is “frames” then the function reports the frames support of the client browser, if the parameter is “applets” then the Java applets support is returned, and if any other option is passed, nothing is returned, which is the same for all requests, thus not leading to a new version of a page being cached each time one is hit that uses a VaryByCustom with a value other than “frames” or “applets”.

This added functionality provides for numerous options as to which pages are cached and which are generated. Assuming that the VaryByCustom attribute has a value of “frames,applets”, there could be, at most, 4 different versions of a page in cache, for these scenarios:

Frames support, applets support

No frames support, no applets support

Frames support, no applets support

No frames support, applets support

The VaryByCustom attribute could also have the values “frames” or “applets”, in which case there is a maximum of 2 cached versions for each, respectively.

There are myriads of possible uses for VaryByCustom, and since the actual criteria is developed exactly as you want it, from all the data available for the request, the likelihood of an output caching scenario made impossible because data would not be able to be properly refreshed practically diminishes. From varying the cache according to the contents of cookies sent in the HTTP request body, to varying by client certificates, VaryByCustom is fit to handle the task.

> As with all things in programming, search for the simplest solution to your problem. As has already been stated, it is only in fairly specific circumstances when using VaryByCustom is actually *required*, versus where it *can* be used. Often the other attributes for the OutputCache directive will offer the functionality that is required, without any need to use custom caching logic.

Programmatic Output Caching

There is an alternative to using the OutputCache directive to implement Output Caching in your pages. The Cache property of the Response object, which is an instance of the HttpCachePolicy class, contains properties that can be used to both enable, and define the criteria for output caching for a page. Unfortunately it is not able to modify all of the properties that the OutputCache directive has access to, however, it can perform most of the functionality of the OutputCache directive, but also has more specific control over the HTTP caching headers, which were discussed in the ‘Location’ attribute section previously. However, one of the major advantages of programmatic output caching is the ability to conditionally set output caching parameters, not easily achievable when using the directive.

Simple Caching

The simplest form of caching using the Cache property that has a direct equivalent to the OutputCache directive makes the page available as a cached version on all nodes (client, downstream and server) with a timeout of 60 seconds. To do this, two methods need to be called – one to define when the cache should expire (similar to the Duration attribute), and another to set the availability of the cached copy. This can be thought of as the Location attribute; however, it must be explicitly set for caching to be enabled, unlike in using the directive, where it can be omitted.

```
Response.Cache.SetExpires(DateTime.Parse(“6:00:00PM”))
Response.Cache.SetCacheability(HttpCacheability.Server)
```

The SetExpires method takes a DateTime as its only parameter. Because of this, it can be used to define absolute expiration times, such as demonstrated above, as well as offer functionality similar to the Duration attribute, which defines the amount of time in seconds that the page will be valid in the cache. The following code demonstrates how to emulate this functionality using SetExpires.

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(60))
Response.Cache.SetCacheability(HttpCacheability.Server)
```

The SetCacheability method takes in a value from the HttpCacheability enumeration in the System.Web namespace. This enumeration contains the values detailed in Table 16.3.

Name	Description
NoCache	Completely disables caching for this page by setting the “Cache-Control: no-cache” header.
Private	Caching is only enabled on the client, and not by proxies.
Public	Caching is enabled on both proxies, and the client.
Server	The response is cached only on the server, and the page is sent with the “Cache-Control: no-cache” header.

[Table 16. 3]

The SetCacheability method and the HttpCacheability enumeration are similar to the Location attribute, and the System.Web.UI.OutputCacheLocation enumeration, although there are differences. The HttpCacheability enumeration deals mainly with the HTTP headers dealing with caching, and only contains one option where the cached copy is located on the server, whereas the OutputCacheLocation enumeration offers slightly different functionality. See Table 16.2 for more details.

Disabling Server Caching

The quickest way to disable caching on the server, although not necessarily on the client and proxies, is to use the SetNoServerCaching method. For example, to have a page cached on proxies and the client, but not on the server, for a time period of 60 seconds, the following code could be used.

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(60))
Response.Cache.SetCacheability(HttpCacheability.Public)
Response.Cache.SetNoServerCaching()
```

It is important to note that once this method has been invoked, it is impossible to re-enable server caching for the current response.

Varying by HTTP Header

The SetVaryByCustom method, despite what its name may suggest, is not the equivalent of the VaryByCustom attribute, but offers functionality more similar to that of the VaryByHeader attribute. Specifically, the SetVaryByCustom method sets the HTTP “Vary” header to the string value that you specify. This ensures that the server responds with a cached copy of the page where the header value of the current request is the same as the cached copy for the specified header. VaryByHeader=”Accept-Charset” would have similar functionality to the following statement.

```
Response.Cache.SetVaryByCustom(“Accept-Charset”)
```

Fragment Output Caching

Fragment, or partial, output caching is the caching of specific parts of a page. There are many situations where fragment caching can be used to effectively reduce the load on the server. It is particularly useful for large pages, where substantial portions of the page need not be regenerated for every page view, but other parts of the page require per-view regeneration, or a much smaller duration for caching, to avoid obsolete information being given. An example of such a situation would be a news portal website. The headlines section and the stock tickers would need to be kept fairly up-to-date, however, the weather forecasts would only need to be updated every 2 hours, at the most. Using regular Output Caching, the “Duration” would have to be one that is suitable for the stocks to be updated in a timely fashion, for example, 5 minutes. This means that every 5 minutes the page would be regenerated, and possibly even more depending on whether any “VaryBy” attributes are used. However, the weather only needs to be updated every 2 hours, so every time there is a hit to the portal’s main page, there is an unnecessary hit to the database to retrieve the weather information. This is unnecessarily wasting server resources, both on the web server, and the database server.

The answer to the problem is fragment output caching. Fragment Output Caching would have a more self-explanatory name if it were called “User Control Output Caching”, as this is essentially what it is. In Chapter 13 on creating user controls, it was said that it is good practice to create reusable, self-contained components to build your application up from. One of the reasons for this is from a maintenance point of view – to follow the news portal example, if the weather ticker required changes, but it was not implemented as a user control, the code would have to be changed on every page on which it was displayed. However, if it were a user control, only one file would need to be changed. One of the many other reasons to make use of user controls is for the purpose of output caching. This is really an extension of the “self-contained unit” definition of user controls. Output Caching User Controls is almost identical to that of normal ASP.NET pages. All that is required is an OutputCache directive. However, the directive offers slightly different attributes to that for regular pages. As such, the general syntax for the OutputCache directive for User Controls is as follows.

```
<%@ OutputCache Duration="#ofseconds" VaryByControl="propertyname"
VaryByParam="parametername" %>
```

The simplest form of caching for user controls is exactly the same as for pages – the OutputCache directive is used, with a value in seconds for the length that the output is to be cached for in the Duration attribute, and a value of “none” in the VaryByParam attribute:

```
<%@ OutputCache Duration="60" VaryByParam="none" %>
```

Listing 16.10 shows an ASP.NET with a user control that has caching enabled. Listing 16.11 is the User Control UI, and Listing 16.12 the User Control logic.

```
<%@ Register TagPrefix="uc1" TagName="FragmentOutputCaching"
Src="FragmentOutputCaching.ascx" %>
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="OutputCachingDemo.aspx.vb"
Inherits="Caching.OutputCachingDemo"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
    <HEAD>
        <title></title>
        <meta name="GENERATOR" content="Microsoft Visual
Studio.NET 7.0">
        <meta name="CODE_LANGUAGE" content="Visual Basic
7.0">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
    </HEAD>
    <body>
        <form id="frmOutputCachingDemo" method="post"
runat="server">
            <uc1:FragmentOutputCaching
id="FragmentOutputCaching1" runat="server">
            </uc1:FragmentOutputCaching>
        </form>
    </body>
</HTML>
```

[Listing 16.10]

```
<%@ Control Language="vb" AutoEventWireup="false"
Codebehind="FragmentOutputCaching.ascx.vb"
Inherits="Caching.FragmentOutputCaching" %>
<%@ OutputCache Duration="60" VaryByParam="none" %>
<asp:Label id="lblGenerated" runat="server">
</asp:Label>
```

[Listing 16.11]

```
Public MustInherit Class FragmentOutputCaching
  Inherits System.Web.UI.UserControl
```

```
    Protected WithEvents lblGenerated As System.Web.UI.WebControls.Label

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

    End Sub

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
        lblGenerated.Text = DateTime.Now.ToString("G")
    End Sub

End Class
```

[Listing 16.12]

The ASP.NET page contains nothing special. It has a user control added the same way any other user control is added, as is detailed in Chapter 13. The user control itself also contains nothing notable, with the exception of the OutputCache directive. The User Control logic also contains nothing to suggest that the user control is cached. Summed up, basic fragment caching is a very simple affair, and only involves adding one directive to the user control UI file.

> Unlike regular output caching, Fragment output caching creates a new version of the cached control for every page or form that it is used on. For example, if you use a user control, with caching enabled, called “mycontrol.ascx” and then use the control on two pages, “page1.aspx” and “page2.aspx”, 2 different versions of “mycontrol.ascx” will be cached.

Caching various different versions of user controls

As with page output caching, fragment output caching allows for the scenario that not all generated code will be the same for each request, and the code that is generated might depend on querystring/POST variables, or more importantly in the case of user controls, property values. The VaryByParam directive is available to vary the cached output versions of user controls by the querystring or POST values that are passed to them, and works identically to the way in which it works for normal output caching. However, Microsoft doesn’t recommend this method of varying the output cache, due to complexities in ASP.NET’s method of transferring querystring and POST variables to user controls. That said, the below line would vary the cached output versions of the user control which it was inserted into according to the “country” parameter value of the page that it is running in.

```
<%@ OutputCache Duration="60" VaryByParam="country" %>
```

The next, and possibly most important, method of varying the cache for different versions of user controls is by properties set by attributes. The primary method of setting initial user control properties is through attributes in the User Control definition tag in the UI of the ASP.NET page. For example, when you set the text property of a Label Web Control, the definition may look like this:

```
<asp:Label id="SomeLabel" runat="server" Text="Label Text" />
```

However, in code, you could also set the property like this:

```
SomeLabel.Text = “Label Text”
```

Fragment caching automatically generates a different version of each user control for every separate instance where any property set using an attribute in the User Control definition tag is different from the properties set in the version(s) that are already cached. Only the properties set using attributes will be taken into account for caching purposes. Properties set or modified programmatically are not automatically used by the caching engine for creating different versions of the user control in the cache.

To demonstrate this concept, Listings 16.13, 16.14 and 16.15 show important sections in an ASP.NET page, a user control UI file and the user control’s code-behind file.

```
<%@ Register TagPrefix="uc1" TagName="FragmentOutputCaching"
Src="FragmentOutputCaching.ascx" %>
...
<uc1:FragmentOutputCaching id="FragmentOutputCaching1" Text="There
is a different version of this control in cache for each different 'Text' property
value." runat="server">
</uc1:FragmentOutputCaching>
```

[Listing 16.13]

```
<%@ OutputCache Duration="60" VaryByParam="none" %>
<%@ Control Language="vb" AutoEventWireup="false"
Codebehind="FragmentOutputCaching.ascx.vb"
Inherits="Caching.FragmentOutputCaching" %>
<P>
        <asp:Label id="lblText" runat="server">
        </asp:Label>
</P>
<P>
```

```
        <asp:Label id="lblGenerated" runat="server">
        </asp:Label>
</P>
```

[Listing 16.14]

```
Private strText As String = ""

Property Text() As String
  Get
    Return strText
  End Get
  Set(ByVal Value As String)
    strText = Value
  End Set
End Property

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  lblText.Text = strText
  lblGenerated.Text = DateTime.Now.ToString("G")
End Sub
```

[Listing 16.15]

If the ASP.NET page is loaded in a browser, there will be two paragraphs displayed, one with the text “There is a different version of this control in cache for each different 'Text' property value” and the second displaying the current date and time. Notice in listing 16.14 that the OutputCache directive does not include any special attributes. Refreshing the page in the browser will (so long as you do it within 60 seconds of loading the page originally) produce exactly the same result. Naturally after the 60 seconds has expired, a new version will be cached, with the updated time displayed. However, if you modify the text attribute in the ASP.NET page and then reload, you will see that an entirely new version of the page is cached. Subsequent requests are then loaded from the cache.

> It is important to note that the Text property need not have been a property in order for the cache to vary by it. If it was actually only a public string, the caching engine would still use it, so long as it is still being set declaratively by the user control tag. I.e. it would still work even if the Text property in the User Control code-behind file was removed and replaced with this (assuming that nothing else is changed):

```
Public String Text
```

VaryByControl

User controls may often contain other web controls themselves, which will of course expose properties of their own. The values of these properties may possibly affect the output of the user control. It is therefore important that different versions of the user control are created for the controls whose properties may affect the output of the control. Hence Microsoft included the VaryByControl attribute of the OutputCache directive for user controls. VaryByControl works very similarly to VaryByParam – it takes a semi-colon separated list of control names, and forces different versions of the user control to be cached where the properties of any of the specified controls are different from the cached version(s)’s controls’ properties. Listing 16.15.1 and 16.15.2 show the UI and code-behind code for a user control that demonstrates this.

```
<%@ Control Language="vb" AutoEventWireup="false"
Codebehind="FragmentOutputCaching.ascx.vb"
Inherits="Caching.FragmentOutputCaching" %>
<%@ OutputCache Duration="10" VaryByControl="cboChoice"
VaryByParam="none"%>
<P>
    <asp:DropDownList id="cboChoice" runat="server">
        <asp:ListItem Value="This is option 1.">
            Option 1</asp:ListItem>
        <asp:ListItem Value="This is option 2.">
            Option 2 </asp:ListItem>
        <asp:ListItem Value="This is option 3.">
            Option 3 </asp:ListItem>
    </asp:DropDownList>
    <asp:Button id="btnGo" runat="server" Text="Go">
    </asp:Button>
</P>
<P>
    <asp:Label id="lblSelected" runat="server">
    </asp:Label>
</P>
```

[Listing 16.15.1]

```
Public MustInherit Class FragmentOutputCaching
  Inherits System.Web.UI.UserControl
  Protected WithEvents cboChoice As
System.Web.UI.WebControls.DropDownList
  Protected WithEvents btnGo As System.Web.UI.WebControls.Button
  Protected WithEvents lblSelected As System.Web.UI.WebControls.Label

#Region " Web Form Designer Generated Code "

  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
```

```
    End Sub

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
    End Sub

    Private Sub btnGo_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGo.Click
        lblSelected.Text = cboChoice.SelectedItem.Value & " This was generated
on " & DateTime.Now.ToString() & "."
    End Sub

End Class
```

[Listing 16.15.2]

To test this sample, insert this user control into a new web form and load it. A listbox should be shown with 3 options. Choose an option and click “Go”. The page will reload, and some text below the listbox should appear saying, “This is option X. This was generated on DATE TIME.” If you choose another option, the date and time will change, but after returning to the original option (if you do so within 10 seconds), the date and time will have remained the same.

The code in Listing 16.15.1 specifies that the control should be cached for 10 seconds, and the output should be varied by the control “cboChoice”. When you choose a new value for the listbox, the properties of it are changed (specifically, the SelectedItem-type properties). This causes a new version of the control to be cached. However, when you return to an option that you have already chosen, the cached version of the control can be retrieved.

Data Caching

Output and fragment caching are a great way to greatly increase the performance of your web applications by holding entire pre-generated versions of pages to be served to users. However, there are many cases where a greater level of detail and control of the cache is required. There are also situations where it would be very advantageous to be able to access the cached information programmatically. Enter

Data Caching. ASP.NET Data Caching gives you access to a Cache object that can be used to store and retrieve pieces of information that you’d like to cache and access programmatically. This object is an instance of the System.Web.Caching.Cache class, and is a property of the Page object. This Cache class is extremely powerful and contains more functionality than any 3rd party component for ASP 3 that I am aware of. In its simplest form, you can use the object to store and retrieve values, much like you would using the Session object. However, also included are the facilities for adding time expirations for items in the cache, dependencies, item priority and a delegate callback when items in the cache expire.

Getting Started with Data Caching

The easiest and quickest form of data caching uses the Cache object almost as if it were the Session object – items are added to and retrieved from the Cache using a key and value pair, in the format Cache(“Key”) = Value. Values can then be retrieved using Cache(“Key”). The Cache class implements the IEnumerable interface, which allows, as with the Session class and many others, to loop through the contents of the Cache object using a For Each loop.

Without further delay, listing 16.16 shows a simple ASP.NET page with a label, and listing 16.17 shows the code-behind for the page. When the page is loaded, a value is inserted into the Cache object, and then immediately retrieved and placed into the label. However, further requests for the page retrieve the value straight from the cache.

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="DataCachingDemo.aspx.vb"
Inherits="Caching.DataCachingDemo"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
    <HEAD>
            <title></title>
            <meta name="GENERATOR" content="Microsoft Visual
Studio.NET 7.0">
            <meta name="CODE_LANGUAGE" content="Visual Basic
7.0">
            <meta name="vs_defaultClientScript" content="JavaScript">
            <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
    </HEAD>
    <body>
            <form id="frmDataCachingDemo" method="post"
runat="server">
                    <asp:Label id="lblCachedValue"
runat="server"></asp:Label>
            </form>
```

```
        </body>
</HTML>
```

[Listing 16.16]

```
Public Class DataCachingDemo
    Inherits System.Web.UI.Page
    Protected WithEvents lblCachedValue As
System.Web.UI.WebControls.Label

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

    End Sub

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
        If Cache("MyKey") = Nothing Then
            Cache("MyKey") = "My Value!"
        End If
        lblCachedValue.Text = Cache("MyKey")
    End Sub

End Class
```

[Listing 16.7]

If you load the page, it should display “My Value!”. Firstly, the page checks to see if the Cache(“MyKey”) object contains anything, and if not, sets the value to “My Value!”. Next, the page sets the text property of the label to Cache(“MyKey”), which will retrieve the value stored under “MyKey” in the data cache.

There are two important pieces of information that I haven’t yet told you. Firstly, as could be logically expected, the Cache object persists over multiple users. In other words, if you store a value in the cache once, all other sessions will be able to access that variable. Therefore, in that regard, the Cache object is very similar to using Application variables. Secondly, unlike with Session variables, it is perfectly acceptable to store objects in the cache. With session variables, storing objects could possibly cause excessive strain on the server, since a new set of session variables is created for every single user. However, with the Cache object, as with the Application object, there's only one set of variables. As such, the actual syntax for simply storing values in the data cache is as follows.

```
Cache("Key") = object
```

You cannot only store string values in cache variables, but another other type of object. You could, for example, quite safely store a dataset. This makes data caching even more useful.

Cache.Add and Cache.Insert

Up until now, the Cache object has provided no features that the Application object doesn't provide equally as well. The Cache("Key") = value method is quick and easy, but if you're looking to really use the full power of the Cache object, then you need to inspect the Add and Insert methods of the Cache object. These methods provide the bulk of the functionality of the Cache object.

Firstly, what's the difference between Add and Insert? Both methods actually do exactly the same thing (i.e. insert an object into the cache), and both take exactly the same parameters. The subtle, yet important difference is that Add requires you to provide values for all the parameters, whilst Insert provides several overloaded versions that are perfect for most situations and don't require unnecessary adding of "Nothing"'s because a required parameter isn't going to be used. For this reason, this chapter will be using Insert sparingly, with the majority of examples using Add, but remember that every example could always be done using Insert.

The most complete (i.e. the version with the maximum number of parameters included) version of the Insert definition reads as follows.

```
Overloads Public Sub Insert( _
  ByVal key As String, _
  ByVal value As Object, _
  ByVal dependencies As CacheDependency, _
  ByVal absoluteExpiration As DateTime, _
  ByVal slidingExpiration As TimeSpan, _
  ByVal priority As CacheItemPriority, _
  ByVal priorityDecay As CacheItemPriorityDecay, _
  ByVal onRemoveCallback As CacheItemRemovedCallback _
)
```

Table 16.4 briefly outlines the purpose of each parameter, so that you can just go to the relevant section in the chapter if you're looking for something specific.

Parameter	Purpose
key	The key used to index the cache item by
value	The object to be stored in the cache

dependencies	The files or other cache keys that this item relies on – if they change, the cache item must be refreshed.
absoluteExpiration	The absolute date and time when the item in the cache will expire and be removed.
slidingExpiration	The time between the last access of the item and when the item will expire.
priority	When the Cache object needs to remove objects, it uses this value to determine how important each item is
priorityDecay	How quickly an item’s priority will decay.
onRemoveCallback	A delegate (or “function pointer”) that will be called when the item is removed from cache.

[Table 16.4]

Cache.Add and Cache.Insert – Simple Usage

Passing only the key and value parameters, Cache.Add and Cache.Insert can be used to obtain exactly the same results as using Cache(“Key”) = Value. When using these methods to add items to the cache, items are still retrieved in exactly the same way – Value = Cache(“Key”). The following line of code shows the simplest use for the Cache.Insert function.

```
Cache.Insert("MyKey", "My Value!")
```

Since the Add method requires all possible parameters to be passed, using it is slightly more cumbersome:

```
Cache.Add("MyKey", "My Value!", Nothing, Cache.NoAbsoluteExpiration,

Cache.NoSlidingExpiration,

System.Web.Caching.CacheItemPriority.Normal,

System.Web.Caching.CacheItemPriorityDecay.Default, Nothing)
```

This line of code passes the key “MyKey” and value “My Value!”, and then proceeds to pass the values that Cache.Insert assumes as defaults if they are not passed. As you can see, if would generally be easier to use the Insert method in this case, as it obtains the same result without you having to bother completing numerous parameters.

Dependencies

Previously one of the largest difficulties in caching was calculating when items in the cache should be invalidated and new items retrieved. Previously a trade-off between time delay length and necessity of up-to-date information was required. Therefore, if a stock broking site was caching the latest prices from an XML file, they may choose to invalidate the cache every minute, as it is essential for the stock quotes to be kept relevant, whilst a news site may only update the cache of its latest stories every 30 minutes, since it’s headlines are unlikely to change largely within a 30 minute period. Wouldn’t it be great if those sites could simply detect when the XML file holding the data changed, and only update the cache then? That way there would be no wasted updates (updates to the cache where nothing has changed), and there would be no delayed information, since as soon as the file changes, the cache is updated – no waiting for the cache expiration timeout to complete.

The 3rd parameter in the Add and Insert methods take care of this problem – the parameter accepts an instance of the CacheDependency class, which specifies either a file, an array of files or an array of files and an array of other Cache keys on which the current entry is dependant. If any of the files or keys specified are modified, the item in the cache is invalidated (it is deleted). Listing 16.18 shows an example of creating a CacheDependency object and specify a file on which the key must be dependant.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  'Put user code to initialize the page here
  If Cache("DependentKey") = Nothing Then
    Cache.Insert("DependentKey", DateTime.Now.ToString("G"), New
System.Web.Caching.CacheDependency(Server.MapPath("datafile.xml")))
  End If
  lblCachedValue.Text = Cache("DependentKey")
End Sub
```

[Listing 16.18]

To try this out for yourself, you need a web form with a label Web Control “lblCachedValue”, and a file “datafile.xml” in the same directory as the ASP.NET page. When you load the page for the first time, the current date and time should be displayed. Since the value is cached, if you reload the page, the date and time value will remain the same. However, if you open the “datafile.xml” file, edit it and save it, then refresh the page, you will see that a new value has been cached.

It is also possible to cause a cache key to be dependent on more than one file. This is achieved using the 2nd overloaded constructor of the CacheDependency class, which takes an array of strings containing the names of files as its only parameter. Continuing from the previous example, you’ll need to add another file, “datafile2.xml” to the application’s directory before testing out the code in Listing 16.19.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  'Put user code to initialize the page here
  Dim arrDependentFiles() As String = {Server.MapPath("datafile.xml"),
Server.MapPath("datafile2.xml")}
  If Cache("DependentKey") = Nothing Then
    Cache.Insert("DependentKey", DateTime.Now.ToString("G"), New
System.Web.Caching.CacheDependency(arrDependentFiles))
  End If
  lblCachedValue.Text = Cache("DependentKey")
End Sub
```

[Listing 16.19]

If either one of the files “datafile.xml” or “datafile2.xml” are edited, then the “MyKey” cached entry is invalidated. Again, you can test this out by loading the page and modifying either file, then reloading the page to see that the cache entry’s value has been updated.

> XML files have been used in these examples to demonstrate how dependencies can be used with data caching because the area where this functionality is bound to be used the most is with database files. XML databases are probably most suited to this application, since their “tables” are generally separated into different physical files, but dependencies could also be used to invalidate the cache based on other database files, such as Microsoft Access. However, since all the tables are included in one file, this may become impractical.

The 3rd overloaded version of CacheDependency requires two arrays to be passed, both string arrays, that contain a list of files and a list of cache keys to depend on respectively. The array of filenames has already been covered, but the cache keys list is an interesting concept. Basically it allows a list of keys to be specified, that if any of them changes, the current cache item is invalidated. This can be particularly useful for caching up-to-date database entries, as one cache key could store when the database table(s) had been last updated, and all the keys holding the actual data would be invalidated and forced to reload their data if that one key changed.

Nevertheless, Listing 16.20 shows a much simpler example demonstrating the concept.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  'Put user code to initialize the page here
  Dim arrDependentFiles() As String = {}
  Dim arrDependentKeys() As String = {"KeyToDependOn"}
  If DateTime.Now.Second Mod 2 = 0 Then
    If Cache("KeyToDependOn") <> True Then
      Cache("KeyToDependOn") = True
    End If
  Else
    If Cache("KeyToDependOn") <> False Then
      Cache("KeyToDependOn") = False
    End If
  End If
  If Cache("DependentKey") = Nothing Then
    Cache.Insert("DependentKey", DateTime.Now.ToString("G"), New
System.Web.Caching.CacheDependency(arrDependentFiles,
arrDependentKeys))
  End If
  lblCachedValue.Text = Cache("DependentKey")
End Sub
```

[Listing 16.20]

This really is a trivial example, but it demonstrates at least one important possible pitfall when using cache key dependencies. The If blocks dealing with the Cache item “KeyToDependOn” check whether the current second is an even or an odd number. If event, “KeyToDependOn” should store True, and if not, False. You may then observe, “Why don’t we just do this then?”

```
If DateTime.Now.Second Mod 2 = 0 Then
```

```
    Cache("KeyToDependOn") = True
  Else
    Cache("KeyToDependOn") = False
  End If
```

The reason is that every time that code is run, a value is always assigned to Cache(“KeyToDependOn”) – either True or False. It doesn’t matter whether the key originally stored the value, it is simply assigned. The problem caused by this is that the cache key dependency functionality assumes that an assignment to a cache key means that it’s content is changing. Therefore, if the If blocks handling the “KeyToDependOn” logic were replaced with this one here, the “DependentKey” item would be updated every time the page was loaded.

With that matter clearup, you can now run the sample. Take note of the time that is displayed and remember whether the seconds are even or odd. When you reload the page, if the time remains the same, you can safely assume that the current second’s even-or-odd status was the same as the previous refresh. However, if the time changed, it obviously was not.

You’re unlikely to ever actually do anything as trivial as this in a real application, but the principles hold true for all scenarios. For example, if you’re using a key to store that last date that a database table was changed, which other keys will depend on, then you can’t simply load and assign the date and time of the most recent record to the cache item – you must first check if it is different, then, if it is, do the assignment.

Expirations

Absolute Expirations

Dependencies can be used to invalidate cache entries according to external factors, such as a file changing. Expirations allow cache entries to be invalidated according to either specific times (Absolute Expirations) or specific time intervals (Sliding Expirations).

Absolute Expirations provide a definite, set date and time when a cache item must be invalidated. This may be something such as “05/05/2002 14:14:00”. However, a much more common situation is the usage of Absolute Expirations with values such as “Current Date & Time + 10 minutes”, causing the cache item to be invalidated 10 minutes from the current time.

The usage for these expirations is very simple, and deals with an overloaded version of Cache.Insert that takes the key, value, cache dependency, absolute expiration and sliding expiration values as parameters. Listing 16.21 shows the usage of absolute expirations without using a cache dependency or a sliding expiration.

```
  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
  System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
    If Cache("AbsoluteExpirationKey") = Nothing Then
      Cache.Insert("AbsoluteExpirationKey", "This key will expire at " &
  DateTime.Now.AddMinutes(1).ToString("G") & ".", Nothing,
  DateTime.Now.AddMinutes(1), Cache.NoSlidingExpiration)
    End If
    lblCachedValue.Text = Cache("AbsoluteExpirationKey")
  End Sub
```

[Listing 16.21]

When the page is loaded, a message is display informing you that in a minute’s time, the key will be invalidated. If you refresh the page within the minute, you will see that the value is, of course, still cached. However, once the minute has elapsed, the key will have been invalidated and refreshing the page will reload the key (and the absolute expiration value) with a new time, one minute from the current.

The .NET documentation provides more detail, but the DateTime class provides numerous methods for adding time to the value, and these include AddDays, AddHours, AddMinutes, AddMonths, AddSeconds and AddYears.

Sliding Expirations

Sliding Expirations differ from Absolute Expirations, in that they allow not for a specific date and time to be set, but an amount of time from the last request to a cache item before it is invalidated. In other words, you can specify a time span of 10 seconds for the parameter in the Insert method, and 10 seconds after the last request for the item from the cache, the item will be invalidated.

> Sliding Expirations may at first appear pointless, since a seemingly identical result can be obtained using Absolute Expirations by setting a value that is for example, the current time plus 10 minutes. However, setting this value and setting a sliding expiration value of 10 minutes produce entirely different results – the absolute expiration invalidates the cache 10 minutes from when the item was created. The sliding expiration invalidates the cache 10 minutes from when the cache was last accessed.

Listing 16.22 shows the use of sliding expirations, with a value of 10 seconds. Therefore, 10 seconds after the last time the entry is accessed, it will be invalidated.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  'Put user code to initialize the page here
  If Cache("SlidingExpirationKey") = Nothing Then
    Cache.Insert("SlidingExpirationKey", DateTime.Now.ToString("G"),
Nothing, Cache.NoAbsoluteExpiration, TimeSpan.FromSeconds(10))
  End If
  lblCachedValue.Text = Cache("SlidingExpirationKey")
End Sub
```

[Listing 16.22]

Besides showing the usage of Sliding Expirations, this example can also be used to demonstrate the important difference between Absolute and Sliding Expirations, as mentioned in the note above. You may wish to repeat the instructions below using an Absolute Expiration with a value of 10 seconds from the current time as well, just to be sure that it actually works as I've described it.

1. Load the page.
2. Take note of the time.
3. Wait 5 seconds, then reload the page.
4. Wait until 11 seconds has elapsed since the time of the first load, then reload the page again. If this was an absolute expiration, the item would've been invalidated, and therefore the time would've changed. However, since the last time the cache item was accessed was in your 2nd load, the cache would only invalidate 15 seconds after the 1st load.
5. Take note of the time from the system clock (not the page, since the value is still cached).
6. Wait until 11 seconds has elapsed since this time, then reload the page – you'll notice that a new cache value has been entered, since the previous one was invalidated.

Using Absolute and Sliding Expirations together

It is possible to provide both an absolute and a sliding expiration value in the Insert or Add method. In this case, whichever expiration is reached first will cause the cache to be invalidated. The use of both absolute and sliding expirations together can therefore be considered an "or" condition for invalidation – "if absolute expiration is met, then invalidate OR if sliding expiration is met, then invalidate."

Cache Item Priority

Items in the cache can potentially use up a fairly large amount of system memory, and therefore the cache is the perfect place for a server to look to free up memory when the system is running low, since cache items are never (well, *should* never) be used to store data that isn't located elsewhere, and they are essentially a mechanism to *temporarily* copy items from the hard disc to memory for faster retrieval. Therefore, other than reducing the caching performance advantage, removing items from the cache should have no adverse effects on the application, whilst enabling the server to regain free memory when required. However, naturally there will be items in the cache that you will prefer to be removed before others, if the system does partially purge the cache to regain memory. These items may be the more infrequently accessed items, or the items that are less hard-drive or network resource intensive to acquire if they are not cached.

The Add and Insert methods allow you to specify how important it is that an item remains in the cache, and therefore in which order the items in the cache will be removed if the cache is partially cleared. This is enabled through the use of the priority parameter, which takes a value from the CacheItemPriority enumeration. Table 16.5 shows the values in this enumeration, in order of priority level.

Value	Description
NotRemovable	The cache item cannot be removed from the cache.

High	The cache item is least likely to be removed from the cache.
AboveNormal	The cache item is less likely to be removed from the cache than those with ‘Normal’ priority, and those with priorities below ‘Normal’.
Normal	This is the default value. The cache item is less likely to be removed than those with ‘BelowNormal’ and ‘Low’ priorities.
BelowNormal	The cache item is more likely to be removed than the ‘Normal’ items, but less likely than ‘Low’ items.
Low	The cache item is the most likely to be removed from the cache.

[Table 16.5 – CacheItemPriority Enumeration]

Listing 16.23 shows an example of how the priority of a cache item might be specified when using the Insert method.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  'Put user code to initialize the page here
  If Cache("PriorityKey") = Nothing Then
    Cache.Insert("PriorityKey", DateTime.Now.ToString("G"), Nothing,
Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration,
System.Web.Caching.CacheItemPriority.High,
System.Web.Caching.CacheItemPriorityDecay.Default, Nothing)
  End If
  lblCachedValue.Text = Cache("PriorityKey")
End Sub
```

[Listing 16.23]

Notice that there is no overloaded version of Insert that can accept only the key, value and priority. The overloaded version that supports cache item priorities is as follows.

```
Overloads Public Sub Insert( _
  ByVal key As String, _
  ByVal value As Object, _
  ByVal dependencies As CacheDependency, _
  ByVal absoluteExpiration As DateTime, _
  ByVal slidingExpiration As TimeSpan, _
  ByVal priority As CacheItemPriority, _
  ByVal priorityDecay As CacheItemPriorityDecay, _
  ByVal onRemoveCallback As CacheItemRemovedCallback _
)
```

This means that even if you only want to specify a few of those parameters, including a priority, you’ll need to provide suitable blank values for the rest, as was shown in Listing 16.23.

Priority Decay

The Insert and Add methods also provide a parameter called “priorityDecay”. This can be particularly useful when used in conjunction with the “priority” parameter in sites that have a high volume of traffic, and allows you to specify how quickly an item’s priority will be downgraded if it is not accessed frequently. For example, if an item has a high priority and a priority decay value of “Slow”, then the item is highly unlikely to be removed from the cache, even if it is accessed infrequently, as its priority will only be downgraded slowly.

Table 16.6 shows the members of the CacheItemPriorityDecay enumeration, from which the priorityDecay parameter value is obtained.

Value	Description
Fast	The priority of the item is downgraded the fastest.
Medium	This is the default. The priority of the item is downgraded faster than “Slow” but slower than “Fast”.
Slow	The priority of the item is downgraded the slowest.
Never	The priority of the item is never downgraded.

[Table 16.6 – CacheItemPriorityDecay enumeration]

The aforementioned overloaded version of the Insert method, or the Add method, is again used to supply the priorityDecay parameter, as shown by Listing 16.24.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
    If Cache("PriorityKey") = Nothing Then
        Cache.Insert("PriorityKey", DateTime.Now.ToString("G"), Nothing,
Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration,
System.Web.Caching.CacheItemPriority.High,
System.Web.Caching.CacheItemPriorityDecay.Slow, Nothing)
    End If
    lblCachedValue.Text = Cache("PriorityKey")
End Sub
```

[Listing 16.24]

This code adds a cache item that has high priority and a slow rate of decay if infrequently accessed.

Cache Removal Callbacks

One of the most exciting features of data caching ASP.NET are callbacks that are triggered when items from the cache are removed. These callbacks are enabled through the use of delegates, which are effectively .NET’s implementation of function pointers. Delegates have a very definite relationship with events, and in the following examples of cache removal callbacks, you’ll see several similarities. However, for our purposes, a delegate allows a method to be passed as a parameter to another method. The method itself is not passed, but rather a delegate, which is a type of variable that points to the method.

In the context of cache removal callbacks, you supply a delegate to the Insert or Add method as the last parameter, and this delegate points to the method that you want to be called when the item is removed from the cache. The process is actually very simple, and the functionality that it provides can be a very powerful tool. Listing 16.25 shows a very simple use of a callback that will be fired immediately.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
  'Put user code to initialize the page here
  Dim onRemove As New
System.Web.Caching.CacheItemRemovedCallback(AddressOf
CacheItemRemoved)
  Cache.Insert("SomeKey", "A value", Nothing, Cache.NoAbsoluteExpiration,
Cache.NoSlidingExpiration,
System.Web.Caching.CacheItemPriority.Default,
System.Web.Caching.CacheItemPriorityDecay.Default, onRemove)
  Cache.Remove("SomeKey")
End Sub

Private Sub CacheItemRemoved(ByVal key As String, ByVal value As Object,
ByVal reason As System.Web.Caching.CacheItemRemovedReason)
  lblMessage.Text = "The item was removed."
End Sub
```

[Listing 16.25]

The onRemove variable is of type CacheItemRemovedCallback. This is a delegate, and a member of the Caching namespace. The definition of this delegate in the namespace specifies what parameters the method that the delegate points to must have, in this case, key, value and reason. Effectively, onRemove is a variable that points to a method with 3 parameters of type “String”, “Object” and “CacheItemRemovedReason” respectively.

In the following line, an item is added to the cache. There is nothing new, except for the inclusion of a value for the CacheItemRemovedCallback parameter, “onRemove”. This tells ASP.NET to call the method that our onRemove delegate points to when this particular item is removed from the cache. In the very next line, the item is removed from the cache, which will result in the method “CacheItemRemoved” being called.

To test this code, you simply need a blank web form, with a label named “lblMessage” on it, and the code in Listing 16.25 in the code behind. The page should output the message “The item was removed.” when you load it.

Practical Callbacks

It’s very unlikely that you’re going to remove an item from the cache immediately after adding it, and in any case, if you did, you wouldn’t need to use a callback to write code to execute when the item was removed. The major advantage of callbacks is that you can have code being called when an item is removed, even when nobody is currently requesting the page where the callback was declared and the method is located – it all happens in the background. One of the side effects of this fact is that unless you manually remove the item immediately after adding it, you can’t directly inform the user using a label, or the like, since it’s not guaranteed that any one particular user will be requesting a page when the method fires. Of course, you wouldn’t normally inform the user that an item has been removed from the cache in the callback method – you’d normally do something along the lines of acquiring the latest version of the data that the item stores and recreating it. But nonetheless, it is important to realise this fact. In order to avoid this pitfall, it is necessary to store the text that you want to output in a variable that can be accessed by all page requests, such as an Application variable, or another Cache item. The value from this variable can then be displayed.

The other important behaviour of callbacks that may not be expected is with regard to the removal of items due to expiration, not direct removal using the Remove() method. If you set an expiration value for a cache item (either Absolute or Sliding) and the item is invalidated, the callback method will not be fired until the minute is up (I.e. the system clock reaches the next minute). This does not apply if the item is manually removed.

Listings 16.26 and 16.27 show the UI and code behind source code for a small web form that can be used to demonstrate cache item removal callbacks by allowing you to view and remove items in the cache, as well as add new items, specifying a key, value and number of seconds from creation till expiration. The code also provides for the user-output pitfall that has just been mentioned.

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="DataCachingDemo.aspx.vb"
Inherits="Caching.DataCachingDemo"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
    <HEAD>
        <title></title>
        <meta content="Microsoft Visual Studio.NET 7.0"
name="GENERATOR">
        <meta content="Visual Basic 7.0"
name="CODE_LANGUAGE">
        <meta content="JavaScript" name="vs_defaultClientScript">
        <meta content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
    </HEAD>
    <body>
        <form id="frmDataCachingDemo" method="post"
runat="server">
            <P>
                <asp:label id="lblMessage"
runat="server"></asp:label>
```

```
            </P>
            <P>
                <asp:ListBox id="lstItems"
runat="server"></asp:ListBox>
            </P>
            <P>
                <asp:Button id="btnRemove" runat="server"
Text="Remove Item"></asp:Button>
                <asp:Button id="btnView" runat="server"
Text="View Item"></asp:Button>
                <asp:Button id="btnRefresh" runat="server"
Text="Refresh"></asp:Button>
            </P>
            <P>
                <asp:TextBox id="txtKey" runat="server"
Width="64px" Height="24px">Key</asp:TextBox>
                <asp:TextBox id="txtValue"
runat="server">Value</asp:TextBox>
                <asp:TextBox id="txtAbsoluteExpiration"
runat="server" Width="31px" Height="24px">10</asp:TextBox>
                <asp:Button id="btnAdd" runat="server"
Text="Add Item"></asp:Button>
            </P>
        </form>
    </body>
</HTML>
```

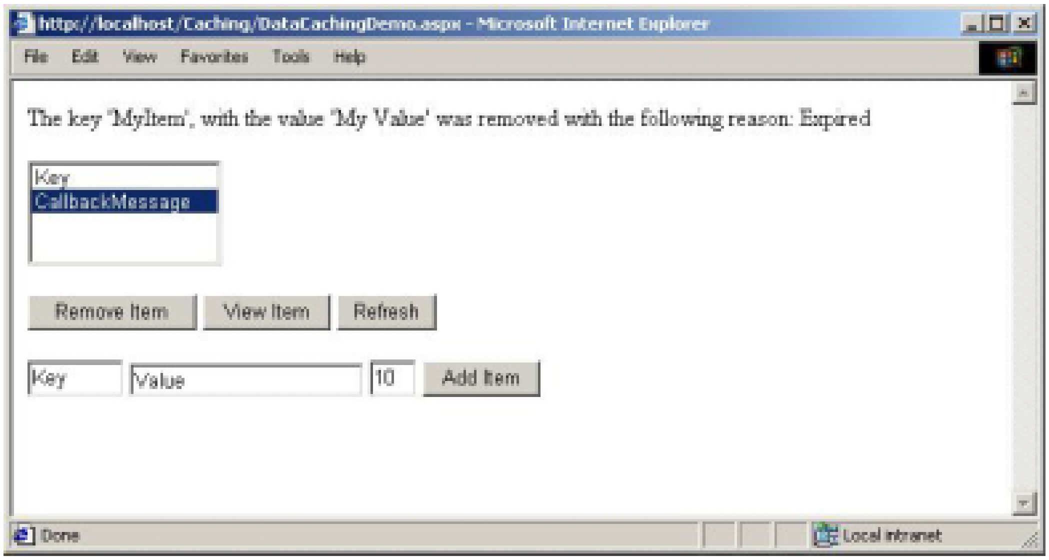
[Listing 16.26]

```
Public Class DataCachingDemo
  Inherits System.Web.UI.Page
  Protected WithEvents lblMessage As System.Web.UI.WebControls.Label
  Protected WithEvents lstItems As System.Web.UI.WebControls.ListBox
  Protected WithEvents btnRemove As System.Web.UI.WebControls.Button
  Protected WithEvents txtKey As System.Web.UI.WebControls.TextBox
  Protected WithEvents btnAdd As System.Web.UI.WebControls.Button
  Protected WithEvents btnView As System.Web.UI.WebControls.Button
  Protected WithEvents txtAbsoluteExpiration As
System.Web.UI.WebControls.TextBox
  Protected WithEvents btnRefresh As System.Web.UI.WebControls.Button
  Protected WithEvents txtValue As System.Web.UI.WebControls.TextBox

#Region " Web Form Designer Generated Code "
```

```
  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

  End Sub

  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub

#End Region

  Private Sub RefreshList()
    lstItems.Items.Clear()
    Dim obj As Object
    For Each obj In Cache
      If Left(obj.Key, 7) <> "System." And Left(obj.Key, 5) <> "ISAPI" Then
        lstItems.Items.Add(obj.Key)
      End If
    Next
  End Sub

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
    If Not Page.IsPostBack Then
      RefreshList()
    End If
    lblMessage.Text = ""
  End Sub

  Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAdd.Click
    Dim onRemove As New
System.Web.Caching.CacheItemRemovedCallback(AddressOf
CacheItemRemoved)
    Cache.Insert(txtKey.Text, txtValue.Text, Nothing,
DateTime.Now.AddSeconds(CInt(txtAbsoluteExpiration.Text)),
Cache.NoSlidingExpiration,
System.Web.Caching.CacheItemPriority.Default,
System.Web.Caching.CacheItemPriorityDecay.Default, onRemove)
        RefreshList()
    End Sub

    Private Sub btnRemove_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRemove.Click
        Cache.Remove(lstItems.SelectedItem.Text)
        RefreshList()
    End Sub

    Private Sub CacheItemRemoved(ByVal key As String, ByVal value As
Object, ByVal reason As System.Web.Caching.CacheItemRemovedReason)
        Cache("CallbackMessage") = "The key '" & key & "', with the value '" &
value & "' was removed with the following reason: " & reason.ToString()
    End Sub

    Private Sub btnView_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnView.Click
        lblMessage.Text = Cache(lstItems.SelectedItem.Text)
    End Sub

    Private Sub btnRefresh_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRefresh.Click
        RefreshList()
    End Sub

End Class
```

[Listing 16.27]

Once you've loaded the form, you can then add, remove and view items. The 1st textbox is for entering a key, the 2nd is the value and the 3rd is the number of seconds after the item has been created that it will expire. Once an item has been removed, either by expiration or by manually removing it, a CallbackMessage item will appear in the list of items in the cache. If you view this, a message will be displayed informing you which item was removed, what its value was and why it was removed. The form may look something like figure 16.3.



The key 'MyItem', with the value 'My Value' was removed with the following reason: Expired

[Figure 16.3 – Web Form using code from listings 16.26 and 16.27]

The RefreshList() method demonstrates how to loop through all the items in the cache using a For Each loop. The If condition in this method checking for key’s starting with “System.” and “ISAPI” is to prevent all the cache items that the system creates from being shown in the listbox.

The CacheItemRemoved() method, which is that the delegate points to, shows how to use the parameters that are passed to it, providing you with the key name and value of the former item, as well as the reason for removal.

The Add button’s click event handler adds items to the cache in the same manner as with the first callback example in Listing 16.25. It uses the user-specified key name and value, has no dependencies, provides an expiration value that will invalidate the key in the number of seconds specified by the user, has default priority and priorityDecay values and specifies a delegate pointing to the CacheItemRemoved() method for the onRemoveCallback parameter.

Conclusion

ASP.NET provides an extremely comprehensive set of functionality that should cater for all caching scenarios. For simpler, more immediate results, output caching can be used to cache pregenerated versions of pages that are regenerated after certain time intervals, and multiple versions of a single page can be cached to allow for numerous different scenarios. Data caching provides an advanced object where items can be stored in memory, rather than requiring roundtrips to database servers or even the local filesystem for each request. This programmatically accessible facility provides greater control over cached items than output caching, but is generally more difficult to implement and can’t be added as an afterthought, as output caching can.