ASP.NET Bible

by mridula Parihar and et al.

ISBN: 0764548166

Hungry Minds © 2002 (855 pages)

Learn the ins and outs of ASP.NET using Visual Basic and C# with this comprehesive reference tool.

Table of Contents

ASP.NET Bible
Preface
Part I - ASP.NET Basics
Chapter 1 - Understanding the .NET Framework
Chapter 2 - Getting Started with ASP.NET
Chapter 3 - Building Forms with Web Controls
Chapter 4 - Using Rich Web Controls
Chapter 5 - Creating and Using Custom Controls
Chapter 6 - Validating User Input
Chapter 7 - Debugging ASP.NET Pages
Part II - ASP.NET Database Programming
Chapter 8 - Introducing ADO.NET
Chapter 9 - Understanding Data Binding
Chapter 10 - Working with Data Grids
Chapter 11 - Using Templates
Chapter 12 - Using SQL Server with ASP.NET
Chapter 13 - Advanced Data Binding and XML
Part III - Advanced ASP.NET
Chapter 14 - ASP.NET Application Configuration
Chapter 15 - Developing Business Objects
Chapter 16 - Building HTTP Handlers
Chapter 17 - Understanding Caching
Chapter 18 - Building Wireless Applications with ASP.NET Mobile Controls
Chapter 19 - ASP.NET Security
Chapter 20 - Localizing ASP.NET Applications
Chapter 21 - Deploying ASP.NET Applications
Part IV - Building Web Services
Chapter 22 - Introduction to Web Services
Chapter 23 - Web Services Infrastructure
Chapter 24 - Understanding SOAP
Chapter 25 - Building a Web Service
Chapter 26 - Deploying and Publishing Web Services
Chapter 27 - Finding Web Services
Chapter 28 - Consuming Web Services
Part V - Building ASP.NET Applications
Chapter 29 - ASP.NET Blackjack
Chapter 30 - Chatty Discussion Forum
Appendix A - Visual Basic Syntax
Appendix B - Visual Basic Functions and Features

Part I: **ASP.NET Basics**

Chapter List

Chapter 1: Understanding the .NET Framework
Chapter 2: Getting Started with ASP.NET
Chapter 3: Building Forms with Web Controls
Chapter 4: Using Rich Web Controls
Chapter 5: Creating and Using Custom Controls
Chapter 6: Validating User Input
Chapter 7: Debugging ASP.Net Pages

Chapter 1: Understanding the .NET Framework

Overview

The Internet revolution of the late 1990s represented a dramatic shift in the way individuals and organizations communicate with each other. Traditional applications, such as word processors and accounting packages, are modeled as stand-alone applications: they offer users the capability to perform tasks using data stored on the system the application resides and executes on. Most new software, in contrast, is modeled based on a distributed computing model where applications collaborate to provide services and expose functionality to each other. As a result, the primary role of most new software is changing into supporting information exchange (through Web servers and browsers), collaboration (through e-mail and instant messaging), and individual expression (through Web logs, also known as Blogs, and e-zines — Web based magazines). Essentially, the basic role of software is changing from providing discrete functionality to providing services.

The .NET Framework represents a unified, object-oriented set of services and libraries that embrace the changing role of new network-centric and network-aware software. In fact, the .NET Framework is the first platform designed from the ground up with the Internet in mind.

This chapter introduces the .NET Framework in terms of the benefits it provides. I present some sample code in Visual C# .NET, Visual Basic .NET, Visual Basic 6.0, and Visual C++; don't worry if you're not familiar with these languages, since I describe in the discussion what each sample does.

Benefits of the .NET Framework

The .NET Framework offers a number of benefits to developers:

- A consistent programming model
- Direct support for security
- Simplified development efforts
- Easy application deployment and maintenance

Consistent programming model

Different programming languages offer different models for doing the same thing. For example, the following code demonstrates how to open a file and write a one-line message to it using Visual Basic 6.0:

```
Public Sub testFileAccess()
   On Error GoTo handle_Error
```

```
    ' Use native method of opening an writing to a file...
    Dim outputFile As Long
    outputFile = FreeFile
    Open "c:\temp\test.txt" For Output As #outputFile
    Print #outputFile, "Hello World!"
    Close #outputFile

    ' Use the Microsoft Scripting Runtime to
    ' open and write to the file...
    Dim fso As Object
    Set fso = CreateObject("Scripting.FileSystemObject")
    Dim outputText As TextStream
    Set outputText = fso.CreateTextFile("c:\temp\test2.txt")
    outputText.WriteLine "Hello World!"
    outputText.Close
    Set fso = Nothing
    Exit Sub

handle_Error:
    ' Handle or report error here
End Sub
```

This code demonstrates that more than one technique is available to create and write to a new file. The first method uses Visual Basic's built-in support; the second method uses the Microsoft Scripting Runtime. C++ also offers more than one way of performing the same task, as shown in the following code:

```
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <stdio.h>

using namespace std;

int main(int argc, char* argv[])
{
    // Use the C Runtime Library (CRT)…
    FILE *testFile;
    if( (testFile  = fopen( "c:\\temp\\test3.txt",
      "wt" )) == NULL ) {
        cout << "Could not open first test file!" << endl;
        return 1;
    }
    fprintf(testFile,"Hello World!\n");
    fclose(testFile);

    // Use the Standard Template Library (STL)...
    ofstream outputStream("c:\\temp\\test4.txt");
    if(!outputStream) {
        cout << "Could not open second test file!" << endl;
        return(1);
    }
    outputStream << "Hello World!" << endl;
    outputStream.close();
    return 0;
}
```

What both code listings demonstrate is that when using different programming languages, a disparity exists among the techniques that developers use to perform the same task. The difference in techniques comes from how different languages interact with and represent the underlying system that applications rely on, thereby increasing the amount of training that developers need. The following code demonstrates how to perform the same tasks in Visual Basic .NET and Visual C# .NET.

Visual Basic .NET:

```
Imports System.IO
Imports System.Text

Module Demo

  Sub Main()
    Dim outputFile As StreamWriter = _
      New StreamWriter("c:\temp\test5.txt")
    outputFile.WriteLine("Hello World!")
    outputFile.Close()
  End Sub

End Module
```

Visual C# .NET:

```
using System.IO;
using System.Text;

class Demo {
  static void Main() {
    StreamWriter outputFile =
      new StreamWriter("c:\\temp\\test6.txt");
    outputFile.WriteLine("Hello World!");
    outputFile.Close();
}
```

The preceding code demonstrates, apart from slight syntactical differences, that the technique for writing to a file in either language is identical — both listings use the `StreamWriter` class to write the "Hello World!" message out to the text files. In fact, unlike the Visual Basic and Visual C++ listings, which demonstrate that there's more than one way to do something *within* the same language, the preceding listings show that there's a unified means of accomplishing the same task by using the .NET Class Library.

The .NET Class Library is a key component of the .NET Framework — it is sometimes referred to as the Base Class Library (BCL). The .NET Class Library contains hundreds of classes you can use for tasks such as the following:

- Processing XML
- Working with data from multiple data sources
- Debugging your code and working with event logs
- Working with data streams and files
- Managing the run-time environment
- Developing Web services, components, and standard Windows applications
- Working with application security
- Working with directory services

The functionality that the .NET Class Library provides is available to all .NET languages, resulting in a consistent object model regardless of the programming language developers use.

Direct support for security

Developing an application that resides on a user's desktop system and uses local resources is easy, from a security point of view, because security simply isn't a consideration in this scenario. Security becomes much more important when you create applications that access data on remote systems or applications that perform privileged tasks on behalf of nonprivileged users, because systems may have to authenticate users, and encryption (scrambling to avoid eavesdropping) may be necessary to secure data communications.

Windows NT, Windows 2000, and Windows XP have a number of security features based on Access Control Lists (ACLs). An ACL contains a number of entries that specify which users may access, or are explicitly denied access, to resources such as files and printers. ACLs are a great way of protecting executable files (applications) from unauthorized access, but they do not secure all parts of the file. The .NET Framework enables both developers and system administrators to specify method-level security. Developers (through easy-to-use programming language constructs called attributes) and systems administrators (by using administrative tools and byediting an application's configuration file) can configure an application's security so that only authorized users can invoke a component's methods.

The .NET Framework uses industry-standard protocols such as TCP/IP and means of communications such as the Extensible Markup Language (XML), Simple Object Access Protocol (SOAP, a standard application messaging protocol), and HTTP to facilitate distributed application communications. This makes distributed computing more secure, because .NET developers cooperate with network connectivity devices as opposed to attempting to work around their security restrictions.

Simplified development efforts

Two aspects of creating Web-based applications present unique challenges to Web developers: visual page design and debugging applications. Visual page design is straightforward when creating static content; however, when you need to present the result of executing a query in a tabular format using an ASP page, page design can get rather involved. This is because developers need to mix traditional ASP code, which represents the application's logic, and HTML, which represents the presentation of the data. ASP.NET and the .NET Framework simplify development by allowing developers to separate an application's logic from its presentation, resulting in an easier-to-maintain code base. ASP.NET can also handle the details of maintaining the state of controls, such as the contents of text boxes, between calls to the same ASP.NET page, thereby reducing the amount of code you need to write. Visual Studio .NET, which is tightly integrated with the .NET Framework, assists developers as they create ASP.NET and other applications by providing visual designers that facilitate visual drag and drop editing, making page layout and form layout a breeze.

Another aspect of creating applications is debugging. Developers sometimes make mistakes; systems don't behave as you expect them to, and unexpected conditions arise — all of these issues are collectively referred to as, using the affectionate term, "bugs." Tracking down bugs — known as "debugging" — quickly and effectively requires developers to be familiar with a variety of tools, sometimes available from a third party, and techniques — a combination of programming techniques and techniques for using a particular tool. The .NET Framework simplifies debugging with support for Runtime diagnostics.

Runtime diagnostics not only help you track down bugs but also help you determine how well your applications perform and assess the condition of your application. The .NET Framework provides three types of Runtime diagnostics:

- Event logging
- Performance counters
- Tracing

Event logging

Windows 2000 and Windows XP have a feature called an Event Log _ a database containing information about important hardware or software events. The Event Log is useful for recording information about the status of your applications and provides systems administrators a means of diagnosing problems, since they can review Event Log entries using the Event Viewer (supplied with Windows and available in the Administrative Tools group in the Control Panel). There are three types of Event Log events:

- **Informational events:** Usually contain basic information, such as an application starting or shutting down
- **Warning events:** Usually provide information about unusual conditions that have the potential to become errors
- **Error events:** Represent critical errors in an application that prevent it from executing normally

Events are stored in Event Logs — Windows supports three types of Event Logs:

- **Application:** Contains messages that applications such as Microsoft SQL Server log
- **System:** Contains messages that device drivers and system services log.
- **Security:** Contains system-generated messages about events that occur when security auditing is enabled

The .NET Framework makes it easy to work with the Event Log as shown in the following code:

```
Imports System
Imports System.Diagnostics


Module eventLogDemo


 Sub Main()
   If Not EventLog.SourceExists("ASPnetBible") Then
     EventLog.CreateEventSource( _
       "ASPnetBible", "Application")
     Console.WriteLine( _
       "Created new EventSource 'ASPnetBible'")
   End If

   Dim evLog As New EventLog()
   evLog.Source = "ASPnetBible"

   ' Note: this listing does not show the
   ' complete message for brevity
   evLog.WriteEntry( "...starting")
   Console.WriteLine("Wrote 'starting'...")

   evLog.WriteEntry("... exiting")
   Console.WriteLine("Wrote 'exit'...")
   End Sub

End Module
```

This code is a Visual Basic .NET console application that creates an Event Source called ASPnetBible and logs the application's starting and exiting events to the system's Application event log — although the listing doesn't show it, both messages are informational.

Performance counters

Performance counters are useful for monitoring the health and performance of an application. You can chart the value of performance counters using the Performance applet in the Administrative Tools folder of the systems Control Panel. The .NET Framework makes it easy for you to read the value of existing performance counters, such as the system's percent CPU Utilization, as well as create your own application-specific performance counters. The following code demonstrates how to work with performance counters in a simple Windows Forms application:

```
' Create a new performace counter
Dim counterCollection As New CounterCreationDataCollection()
Dim couterItem As New CounterCreationData()

counterName = "demoCounter"
perfCatName = "ASPnetBible"

couterItem.CounterName = counterName
couterItem.CounterType =
  PerformanceCounterType.NumberOfItems32
counterCollection.Add(couterItem)
PerformanceCounterCategory.Create(perfCatName, _
"sample counter", counterCollection)

' ...elsewhere in the application - Increment the counter
```

```
Dim perfCounter As PerformanceCounter
perfCounter = New PerformanceCounter()
perfCounter.CategoryName = perfCatName
perfCounter.CounterName = counterName
perfCounter.ReadOnly = False
perfCounter.IncrementBy(50)
System.Threading.Thread.Sleep(2000)
perfCounter.IncrementBy(-50)

'...elsewhere in the application - Delete the sample counter
PerformanceCounterCategory.Delete(perfCatName)
```

This code demonstrates how to create a new performance counter category and counter using the `CouterCreationDataCollection` and `CouterCreationData` classes — the fragment shown is from the sample application's `Load` event handler. In the next section of the code, from a button's `Click` event handler, the code creates an instance of the sample performance counter, increments it, and waits two seconds before decrementing the counter. The last part of the code shows how to delete the performance counter when the form closes.

Tracing

Debugging an application by using the Visual Studio .NET debugger is a great way to track down problems; however, there are many scenarios in which things happen too quickly to follow interactively or in which you simply need to know the sequence of events that lead to a problem before the problem occurs.

Tracing is an alternative to using a debugger to step through each line of code as your application executes. You can configure ASP.NET tracing by using two methods: page-level tracing and application-level tracing. Both types of tracing provide similar results; however, the difference is in how you access the results for each approach. Page-level tracing provides trace details on the ASPX page when it completes executing, and application-level tracing stores the details of the trace in a file called (by default) `trace.acx`, which is located in the same directory as the ASP.NET application — you can view the file by using your browser.

When you enable tracing, which is disabled by default, ASP.NET records detailed information about the page request, trace messages, control information, cookies, header information, the contents of any form fields, and a raw output of the contents of server variables (like `CONTENT_TYPE` and `HTTP_REFERRER`). Table 1-1 shows a fragment of a trace output from a simple ASP.NET page.

Table 1-1: Fragment of an ASP.NET Page Trace

Category	Message	From First(s)	From Last(s)
`aspx.page`	`Begin Init`		
`aspx.page`	`End Init`	0.000096	0.000096
`aspx.page`	`Begin LoadViewState`	0.000189	0.000092
`aspx.page`	`End LoadViewState`	0.000308	0.000119
`aspx.page`	`Begin ProcessPostData`	0.000393	0.000085

Table 1-1: Fragment of an ASP.NET Page Trace

Category	Message	From First(s)	From Last(s)
`aspx.page`	`End ProcessPostData`	0.000551	0.000158
	`Page_Load event handler started`	0.000647	0.000096
	`Page_Load event handler exit`	0.000729	0.000082

The last two entries in Table 1-1 are custom Trace messages written using the `Page.Trace.Write(...)` method.

Easy application deployment and maintenance

Applications are often made up of several components:

- Web pages
- Windows forms-based components
- Web services
- Components housed in DLLs

The .NET Framework makes it possible to install applications that use some or all of these components without having to register DLLs (using `regsvr32.exe`) or to create Registration Database (also known as the system Registry) entries.

The .NET Framework makes it easy to deploy applications using zero-impact installation — often all that's required to install an application is to copy it into a directory along with the components it requires. This is possible because the .NET Framework handles the details of locating and loading components an application needs, even if you have several versions of the same component available on a single system. All of this is possible because the .NET Framework records extra information about an application's components — the extra information is called metadata. A component of the .NET Framework, the Class Loader, inspects an application's metadata and ensures that all of the components the application depends on are available on the system before the application begins to execute. This feature of the .NET Framework works to isolate applications from each other despite changes in system configuration, making it easier to install and upgrade applications.

Once an application is running on a system, it is sometimes necessary to change certain traits of the application, such as its security requirements, optional parameters, and even database connections. .NET Framework applications use a configuration model based on application-configuration files. A configuration file is a text file that contains XML elements that affect the behavior of an application. For example, an administrator can configure an application to use only a certain version of a component the application relies on, thereby ensuring consistent behavior regardless of how often the component is upgraded. The following code shows an ASP.NET's basic configuration file; the file is called `web.config`:

```
<configuration>
  <system.web>
    <pages
      buffer="true"
      enableSessionState="true" />
    <appSettings>
     <add key="dsn" value="localhost;uid=sa;pwd="/>
    </appSettings>
  </system.web>
</configuration>
```

This code shows that the ASP.NET application will have page buffering on (pages will be sent to clients only when the page is completely rendered), and that ASP.NET will track individual clients' session information (as shown in the `pages` tag). This code also demonstrates how to define a custom configuration key, called `dsn` — within the `appSettings` section, which applications have access to through the `TraceSwitch` class.

Elements of the .NET Framework

The .NET Framework consists of three key elements (as shown in Figure 1-1):

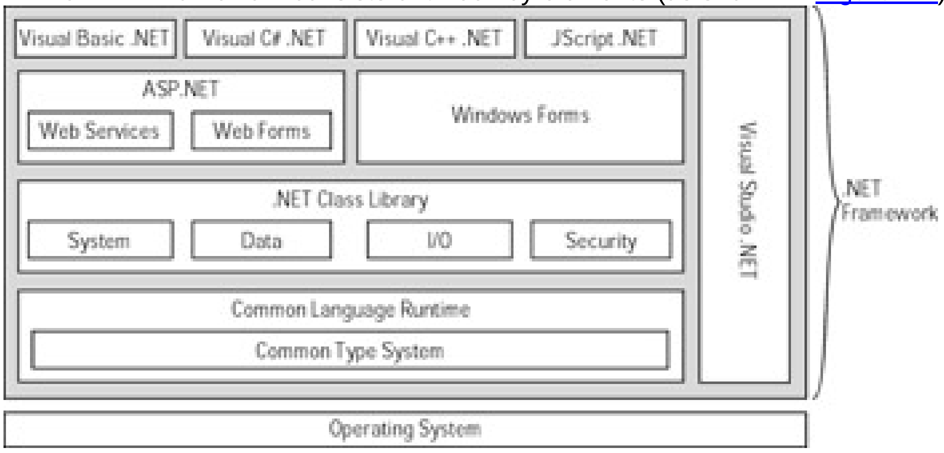


Figure 1-1: Components of the .NET Framework

- Common Language Runtime
- .NET Class Library
- Unifying components

Common Language Runtime

The Common Language Runtime (CLR) is a layer between an application and the operating system it executes on. The CLR simplifies an application's design and reduces the amount of code developers need to write because it provides a variety of execution services that include memory management, thread management, component lifetime management, and default error handling. The key benefit of the CLR is that it transparently provides these execution services to all applications, regardless of what programming language they're written in and without any additional effort on the part of the developer.

The CLR is also responsible for compiling code just before it executes. Instead of producing a binary representation of your code, as traditional compilers do, .NET compilers produce a representation of your code in a language common to the .NET Framework: Microsoft Intermediate Language (MSIL), often referred to as IL. When your code executes for the first time, the CLR invokes a special compiler called a Just In Time (JIT) compiler, which transforms the IL into executable instructions that are specific to the type and model of your system's processor. Because all .NET languages have the same compiled representation, they all have similar performance characteristics. This means that a program written in Visual Basic .NET can perform as well as the same program written in Visual C++ .NET. (C++ is the language of choice for developers who need the best possible performance a system can deliver.)

Common Type System

The Common Type System (CTS) is a component of the CLR and provides a common set of data types, each having a common set of behaviors. In Visual Basic, for example, the String data type maps to the CTS `System.String` class. Therefore, if a JScript .NET client needs to communicate with a component implemented in VB .NET, the client doesn't have to do any additional work to exchange information because it's using a type common to both JScript .NET and VB .NET. The CTS eliminates many interoperability problems that exist outside .NET.

.NET programming languages take advantage of the CTS by enabling developers to use their language's built-in data types — the .NET compilers convert the native data types' into their equivalent CTS types at compile time. Developers can also use CTS types directly in their code if they wish. Table 1-2 describes each standard CTS data type.

Table 1-2: Common Type System Data Types

Complete Name	Description
`System.Byte`	Unsigned 8-bit integer ranging in value from 0 to positive 255
`System.Int16`	Signed 16-bit integer capable of holding values from negative 32,768 to positive 32,767
`System.Int32`	Signed 32-bit integer having a range from negative 2,147,483,648 to positive 2,147,483,647
`System.Int64`	Signed 64-bit integer ranging from negative 9,223,372,036,854,755, 808 to positive 9,223,372,036,854,755, 807
`System.Single`	Single-precision 32-bit floating-point number
`System.Double`	Double-precision 64-bit floating-point number
`System.Decimal`	Signed 96-bit floating-point value with up to 28 digits on either side of the decimal
`System.Char`	16-bit Unicode character (unsigned values)
`System.String`	Sequence of Unicode characters with a capacity of about two billion characters
`System.Object`	32-bit address, referencing an instance of a class
`System.Boolean`	Unsigned 32-bit number that may contain only 0 (False) or 1 (True)

You can use other non-CTS-compliant data types in your applications and components; you're free to use non-CTS-compliant data types, but they may not be available on other implementations of the .NET Framework for other operating systems (see Table 1-3).

Table 1-3: Non-CTS-compliant Data Types

Complete Name	Description
`System.SByte`	Signed 8-bit integer ranging from negative 128 to positive 127
`System.UInt16`	16-bit unsigned integer ranging from 0 to positive 65,535
`System.UInt32`	32-bit unsigned integer ranging from 0 to positive 4,294,967,295
`System.UInt64`	64-bit unsigned integer ranging from 0 to positive 184,467,440,737,095,551, 615

.NET Class Library

In an earlier section, "Consistent programming models across programming languages," the .NET Class Library was described as containing hundreds of classes that model the system and services it provides. To make the .NET Class Library easier to work with and understand, it's divided into *namespaces*. The root namespace of the .NET Class Library is called System, and it contains core classes and data types, such as `Int32`, `Object`, `Array`, and `Console`. Secondary namespaces reside within the System namespace.

Examples of nested namespaces include the following:

- **System.Diagnostics**: Contains classes for working with the Event Log
- **System.Data**: Makes it easy to work with data from multiple data sources (`System.Data.OleDb` resides within this namespace and contains the ADO.NET classes)
- **System.IO**: Contains classes for working with files and data streams

Figure 1-2 illustrates the relationship between some of the major namespaces in the .NET Class Library.

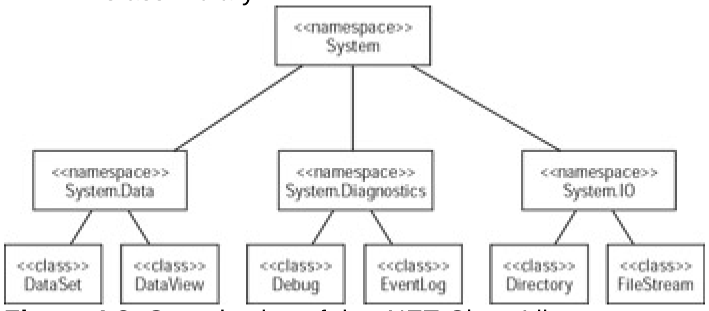


Figure 1-2: Organization of the .NET Class Library

The benefits of using the .NET Class Library include a consistent set of services available to all .NET languages and simplified deployment, because the .NET Class Library is available on all implementations of the .NET Framework.

Unifying components

Until this point, this chapter has covered the low-level components of the .NET Framework. The *unifying components*, listed next, are the means by which you can access the services the .NET Framework provides:

- ASP.NET

- Windows Forms
- Visual Studio .NET

ASP.NET

ASP.NET introduces two major features: Web Forms and Web Services.

Web Forms

Developers not familiar with Web development can spend a great deal of time, for example, figuring out how to validate the e-mail address on a form. You can validate the information on a form by using a client-side script or a server-side script. Deciding which kind of script to use is complicated by the fact that each approach has its benefits and drawbacks, some of which aren't apparent unless you've done substantial design work. If you validate the form on the client by using client-side JScript code, you need to take into consideration the browser that your users may use to access the form. Not all browsers expose exactly the same representation of the document to programmatic interfaces. If you validate the form on the server, you need to be aware of the load that users might place on the server. The server has to validate the data and send the result back to the client. *Web Forms* simplify Web development to the point that it becomes as easy as dragging and dropping controls onto a designer (the surface that you use to edit a page) to design interactive Web applications that span from client to server.

Web Services

A *Web service* is an application that exposes a programmatic interface through standard access methods. Web Services are designed to be used by other applications and components and are not intended to be useful directly to human end users. Web Services make it easy to build applications that integrate features from remote sources. For example, you can write a Web Service that provides weather information for subscribers of your service instead of having subscribers link to a page or parse through a file they download from your site. Clients can simply call a method on your Web Service as if they are calling a method on a component installed on their system — and have the weather information available in an easy-to-use format that they can integrate into their own applications or Web sites with no trouble.

Windows Forms

Windows Forms is the name of a unified set of classes that provides support for creating traditional desktop applications — applications that have a graphical user interface (GUI). Windows Forms make it easy to develop end-user applications using any .NET programming language. Furthermore, through Visual Studio .NET, developers can easily design forms by using drag-and-drop editing.

Visual Studio .NET

Visual Studio .NET fulfills the promise of a single development environment for all languages. Visual Studio .NET simplifies development in a mixed-language environment through features such as support for end-to-end debugging across all programming languages; visual designers for XML, HTML, data, and server-side code; and full IntelliSense support (statement completion). Visual Studio .NET replaces the Visual Basic 6, Visual C++, and Visual InterDev development environments.
Visual Studio .NET is able to provide this level of integration because it relies and builds on the facilities of the .NET Framework. Designers for Web forms and Windows Forms enhance developer productivity during the development cycle. Integration of deployment features enhances productivity during post-deployment debugging. Table 1-4 summarizes Visual Studio .NET's major features.

Table 1-4: Visual Studio .NET's Major Features

Feature	Benefit

Table 1-4: Visual Studio .NET's Major Features

Feature	Benefit
Single IDE	Simplifies mixed-language development with support for Visual Basic, C++, C#, and JScript .NET
Task List	Organizes tasks and manages errors and warnings in a single place. Tasks are read from specialized comments in source code and are presented in a tabular format. Double-click the task to jump to the section of source code where the task was entered.
Solution Explorer	Provides a hierarchical view of a solution organized into projects. Allows the management of related projects within a single solution.
Server Explorer	Manages your computer and other computers on the network, including resources such as SQL Server, message queues, services, and so on. Integrates performance and event monitoring and Web services.
Multi-Monitor support	Makes the best possible use of available screen space
IntelliSense	Ensures consistent statement completion across all supported languages
Dynamic Help	Makes reference documentation available based on what you're working on
End-to-end debugging	Facilitates cross-language, process, and system debugging through the Visual Studio .NET debugger; the learning curve is reduced, and developers are better able to take advantage of the debugger's features.
Deployment support	Integrates deployment into each solution (project); as changes are made in the solution, deployment information is updated. You can deploy your solution using traditional setup (install on a single system), Web setup, and Web download. This feature also facilitates deployment for debugging across systems.

Summary

This chapter introduced you to the .NET Framework and its components in the context of the problems and the benefits the .NET Framework provides. The next chapter discusses setting up the development environment for creating ASP.NET applications, creating a simple ASP.NET application using both VB and C#, and deploying an application on a Web server.

Chapter 2: Getting Started with ASP.NET

Overview

The Microsoft .NET Framework provides a powerful platform for the development of applications for both the desktop and the Internet. The .NET Framework allows you to develop Internet applications with an ease that was never provided before. To develop Internet applications, the .NET Framework is equipped with ASP.NET. ASP.NET is a powerful programming framework for the development of enterprise-class Web applications.

This chapter introduces you to the .NET Framework and ASP.NET. You'll learn to set up the development environment for creating ASP.NET applications. You'll also learn how to create an ASP.NET application by using Visual Basic .NET and C#, and deploy the application.

Introducing the .NET Framework

Since 1995, Microsoft has been constantly making efforts to shift focus from the Windows-based platforms to the Internet. Microsoft introduced Active Server Pages (ASP) as an endeavor toward Internet programming. However, writing ASP script, an interpreted script, was a traditional way of programming as compared to the existing structured object-oriented programming. Moreover, it was very difficult to debug and maintain the unstructured ASP code. Definitely, you could combine the code written in structured object-oriented languages, such as Visual Basic, with ASP code. However, you could combine the VB code only as a component. Moreover, the software integration for the Web development was quite complicated and required an understanding of a host of technologies and integration issues on the part of the developers. Therefore, an architecture was needed that would allow the development of Web applications in a more structured and consistent manner.

Recently, Microsoft introduced the .NET Framework with a vision for developers to create globally distributed software with Internet functionality and interoperability. The .NET Framework includes multiple languages, class libraries, and a common execution platform. In addition, the .NET Framework includes protocols that allow developers to integrate software over the Internet and the .NET Enterprise Servers, such as SQL Server 2000, Commerce Server 2000, and BizTalk Server. Thus, the .NET Framework provides the richest built-in functionality for software integration ever provided by any platform. Also, with the .NET Framework, developing the Internet applications is as easy as developing desktop applications.
The .NET Framework frees the software developer from most of the operating system specifics, such as memory management and file handling, because the .NET Framework covers all the layers of software development above the operating system. Figure 2-1 describes the different components of the .NET Framework.

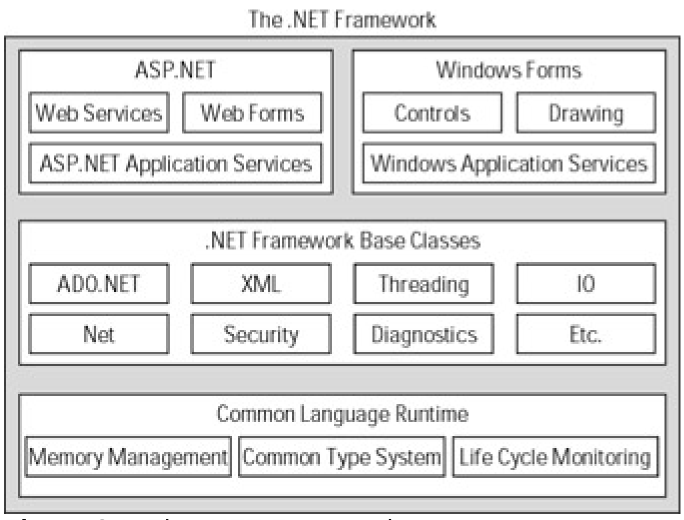


Figure 2-1: The .NET Framework

The top layer represents user and program interfaces, and includes Windows Forms, Web Forms, Web Services, and Application Services. Windows Forms provide a Win32-based user interface. Web Forms provide a Web-based user interface. Web Services are the most revolutionary program interfaces because they allow programs to communicate over the Internet. The Internet-based program interfaces, which include Web Forms and Web Services, are implemented by ASP.NET, which is a built-in component of the .NET Framework.

The middle layer represents the .NET Framework classes, which are universally available across multiple languages. The usage of these classes is consistent across all languages included in the .NET Framework.

The base layer represents the common execution platform called the Common Language Runtime (CLR). This is the most important component of the .NET Framework. The CLR provides support for multiple languages and allows cross-language inheritance. For example, you can inherit a class written in Visual Basic from a class written in Visual C++. Thus, with .NET, the choice of a programming language simply depends on the user's choice. With .NET, it is possible to create applications by using multiple languages. The multiple-language support is possible because the CLR provides a common system of data types. In addition, the CLR performs the memory management and monitors the complete life cycle of objects, while it tracks objects and handles garbage collection.

Visual Studio .NET (VS.NET) is the first release of the products based on the .NET Framework. It includes Visual Basic, Visual C++, and C#. VS.NET provides a common Integrated Development Environment (IDE) for all languages. Therefore, developers always work in a consistent environment irrespective of the language they use.

With that basic understanding of the .NET Framework, you are ready to look at the basic features of ASP.NET.

Introducing ASP.NET

ASP.NET, the next version of ASP, is a programming framework that is used to create enterprise-class Web applications. The enterprise-class Web applications are accessible on a global basis, leading to efficient information management. However, the advantages that ASP.NET offers make it more than just the next version of ASP.

ASP.NET is integrated with Visual Studio .NET, which provides a GUI designer, a rich toolbox, and a fully integrated debugger. This allows the development of applications in a

What You See is What You Get (WYSIWYG) manner. Therefore, creating ASP.NET applications is much simpler.
Unlike the ASP runtime, ASP.NET uses the *Common Language Runtime (CLR)* provided by the .NET Framework. The CLR is the .NET runtime, which manages the execution of code. The CLR allows the objects, which are created in different languages, to interact with each other and hence removes the language barrier. CLR thus makes Web application development more efficient.

In addition to simplifying the designing of Web applications, the .NET CLR offers many advantages. Some of these advantages are listed as follows.

- **Improved performance:** The ASP.NET code is a compiled CLR code instead of an interpreted code. The CLR provides just-in-time compilation, native optimization, and caching. Here, it is important to note that compilation is a two-stage process in the .NET Framework. First, the code is compiled into the Microsoft Intermediate Language (MSIL). Then, at the execution time, the MSIL is compiled into native code. Only the portions of the code that are actually needed will be compiled into native code. This is called Just In Time compilation. These features lead to an overall improved performance of ASP.NET applications.
- **Flexibility:** The entire .NET class library can be accessed by ASP.NET applications. You can use the language that best applies to the type of functionality you want to implement, because ASP.NET is language independent.
- **Configuration settings:** The application-level configuration settings are stored in an Extensible Markup Language (XML) format. The XML format is a hierarchical text format, which is easy to read and write. This format makes it easy to apply new settings to applications without the aid of any local administration tools.
- **Security:** ASP.NET applications are secure and use a set of default authorization and authentication schemes. However, you can modify these schemes according to the security needs of an application.

In addition to this list of advantages, the ASP.NET framework makes it easy to migrate from ASP applications.

Before you start with your first ASP.NET application, take a quick look at how to set up the development environment, described next.

Setting Up the Development Environment

ASP.NET is based on the CLR, class libraries, and other tools integrated with the Microsoft .NET Framework. Therefore, to develop and run the ASP.NET applications, you need to install the .NET Framework. The .NET Framework is available in two forms:

- .NET Framework SDK (Software Development Kit)
- Visual Studio .NET (VS.NET)

You can install the .NET Framework SDK or VS.NET on a machine that has one of the following operating systems:

- Windows 2000
- Windows NT 4.0
- Windows Me
- Windows 98
- Windows XP Professional

Installation of the .NET Framework SDK is very simple — just run the Setup program and follow the onscreen instructions. However, the development machine must have Internet Explorer 5.5 or higher available before the installation. Otherwise, you will be prompted to download it before you install the .NET Framework SDK.

To develop any Web application, you need Internet Information Server (IIS) configured on either the development machine (in the case of Windows 2000 or Windows NT 4.0) or another machine on the network. In the latter case, the .NET Framework must be installed on the machine on which IIS is configured.

Note In the case of Windows 2000 Server, the IIS server is automatically installed.

In addition to installing IIS, you need to install SQL Server 7.0 or higher to develop ASP.NET database applications. You can install SQL Server on the development machine or any other machine on the network.

You can create ASP.NET applications by just installing the .NET Framework SDK and configuring an IIS server. In this case, you need to use a text editor, such as Notepad, to write the code. Therefore, if you do this, you'll have to work without the IDE and other integrated tools that come with VS.NET. Hence, installing VS.NET is recommended, to get the full benefit of the .NET features.

VS.NET Beta 2 comes with four CD-ROMs:

- Windows Component Update CD
- VS.NET CD1
- VS.NET CD2
- VS.NET CD3

When you run the Setup program from VS.NET CD1, a dialog box appears that prompts you for the following three options:

- Windows Component Update
- Install Visual Studio .NET
- Check for Service Releases

If you have not run the Setup program from the Windows Component Update CD, only the first of the preceding three options will be available. In this case, you need to insert the Windows Component Update CD in the CD-ROM drive of the machine and click the first link, Windows Component Update, to begin the update. This option updates Windows with the components that are required to install .NET. Some of the components include Microsoft Windows Installer 2.0, Setup Runtime Files, and Microsoft Data Access Components 2.7. Then, follow the onscreen instructions. In the process, you'll need to reboot the machine several times. After the Windows Component Update is complete, you can use the second link to install VS.NET. After VS.NET is installed, you can click the third link to check for any updates.
When you start Visual Studio .NET, the Start Page is displayed prominently in the window. Figure 2-2 displays the Visual Studio .NET window.

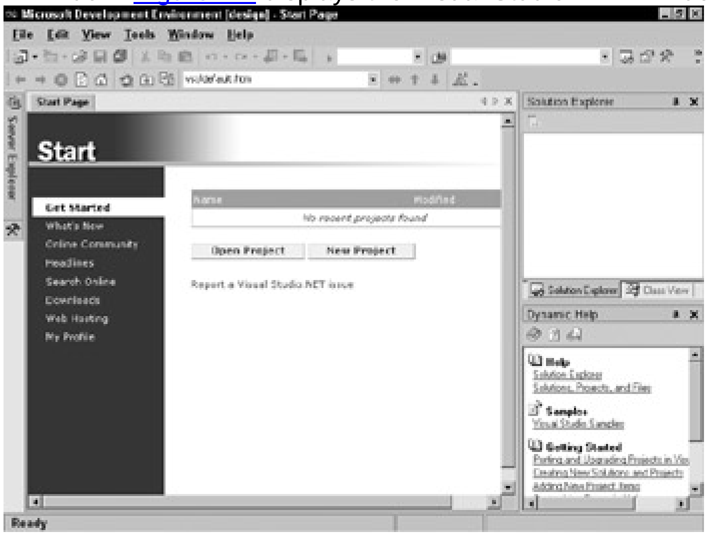


Figure 2-2: The Visual Studio .NET window

The VS.NET window contains the Solution Explorer window to the right. This window displays the projects that are created. The Toolbox and Server Explorer windows can be seen hidden at the extreme left. When you point to the Toolbox or Server Explorer, the corresponding window opens. You use the Toolbox to create the user interface for an application. The Server Explorer window is used to add any Web server or database connection.

The main Start Page window is the central location from where you can perform several tasks, such as create a new project, open an existing project, and get the latest news and recent articles at the MSDN online library. The different options available on the Start Page and what they enable you to do are described as follows:

- **Get Started:** Create a new project or open an existing project.
- **What's New:** Identify the new features of Visual Studio .NET.
- **Online Community:** Contact other developers online. To do this, you must have a newsgroup reader configured on your machine.
- **Headlines:** Get the latest news from the MSDN online library.
- **Search Online:** Search the Web.
- **Downloads:** Get the latest product updates, SDK releases, and sample code from the Internet.
- **Web Hosting:** Post your Web applications and Web services created in Visual Studio .NET directly to the Internet.
- **MyProfile:** Set the IDE-specific preferences.

Creating an ASP.NET Application

After you've set up the development environment for ASP.NET, you can create your first ASP.NET Web application. You can create an ASP.NET Web application in one of the following ways:

- **Use a text editor:** In this method, you can write the code in a text editor, such as Notepad, and save the code as an ASPX file. You can save the ASPX file in the directory C:\inetpub\wwwroot. Then, to display the output of the Web page in Internet Explorer, you simply need to type `http://localhost/<filename>.aspx` in the Address box. If the IIS server is installed on some other machine on the network, replace "localhost" with the name of the server. If you save the file in some other directory, you need to add the file to a virtual directory in the Default WebSite directory on the IIS server. You can also create your own virtual directory and add the file to it.
- **Use the VS.NET IDE:** In this method, you use the IDE of Visual Studio .NET to create a Web page in a WYSIWYG manner. Also, when you create a Web application, the application is automatically created on a Web server (IIS server). You do not need to create a separate virtual directory on the IIS server.

From the preceding discussion, it is obvious that the development of ASP.NET Web applications is much more convenient and efficient in Visual Studio .NET.
ASP.NET Web pages consist of HTML text and the code. The HTML text and the code can be separated in two different files. You can write the code in Visual Basic or C#. This separate file is called the *code behind file*. In this section, you'll create simple Web pages by using VB as well as C#.

Cross-Reference	For more information on code behind files, refer to Chapter 3.

Before you start creating a Web page, you should be familiar with basic ASP.NET syntax. At the top of the page, you must specify an `@ Page` directive to define page-specific attributes, such as language. The syntax is given as follows:

```
<%@ Page attribute = value %>
```

To specify the language as VB for any code output to be rendered on the page, use the following line of code:

<%@ Page Language = "VB" %>

This line indicates that any code in the block, <% %>, on the page is compiled by using VB.

To render the output on your page, you can use the `Response.Write()` method. For example, to display the text "hello" on a page, use the following code:

<% Response.Write("Hello") %>

Note The syntax used in the block, <% %>, must correspond to the language specified in the @ Page directive. Otherwise, an error is generated when you display the page in a Web browser.

You can use HTML tags in the argument passed to the `Response.Write()` method. For example, to display the text in bold, you use the following code:

<% Response.Write("<B> Hello </B>") %>

For dynamic processing of a page, such as the result of a user interaction, you need to write the code within the `<Script>` tag. The syntax of the `<Script>` tag is given as follows:

```
<Script runat="server" [language=codelanguage]>
  code here
</Script>
```

In this syntax . . .

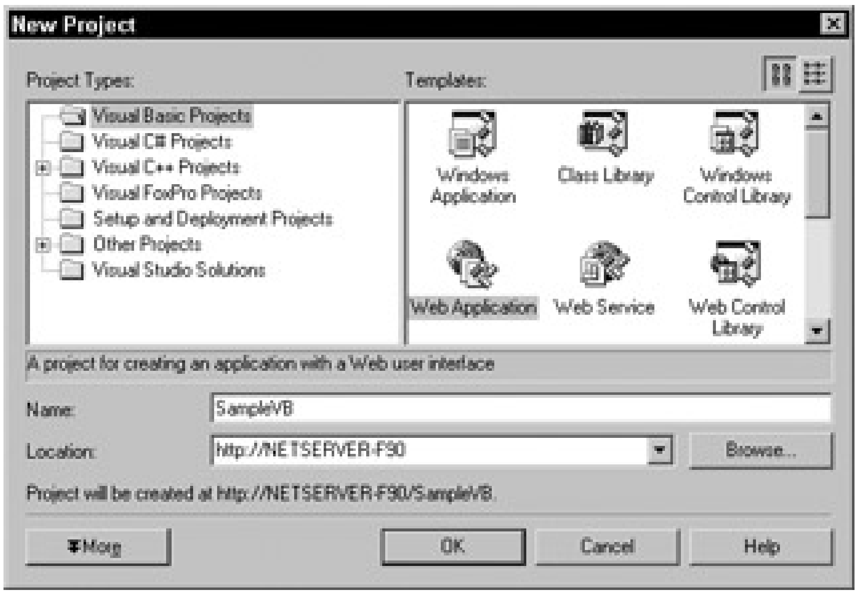
- `runat="server"` indicates that the code is executed at the server side.
- `[language=codelanguage]` indicates the language that is used. You can use VB, C#, or JScript .NET. The square brackets indicate that this attribute is optional. If you do not specify this attribute, the default language used is VB.

After gaining an understanding of the basic ASP.NET page syntax, you can now create a simple ASP.NET Web application. In the following sections, you'll create a simple Web application by using VB and C#. To do so, you'll use the VS.NET IDE.

Creating a Visual Basic Web Application

You can create an ASP.NET application using Visual Basic by creating a Visual Basic Web Application project. To do so, complete the following steps:

1. Select File → New → Project. The New Project dialog box appears.
2. Select Visual Basic Projects from the Project Types pane.
3. Select ASP.NET Web Application from the Templates pane. The Name box contains a default name of the application. The Location box contains the name of a Web server where the application will be created. However, you can change the default name and location. In this case, the name of the sample application is SampleVB. The New Project dialog box now appears as shown in Figure 2-3.


New Project

Project Types:

- Visual Basic Projects
- Visual C# Projects
- Visual C++ Projects
- Visual FoxPro Projects
- Setup and Deployment Projects
- Other Projects
- Visual Studio Solutions

Templates:

- Windows Application
- Class Library
- Windows Control Library
- Web Application
- Web Service
- Web Control Library

A project for creating an application with a Web user interface

Name: SampleVB

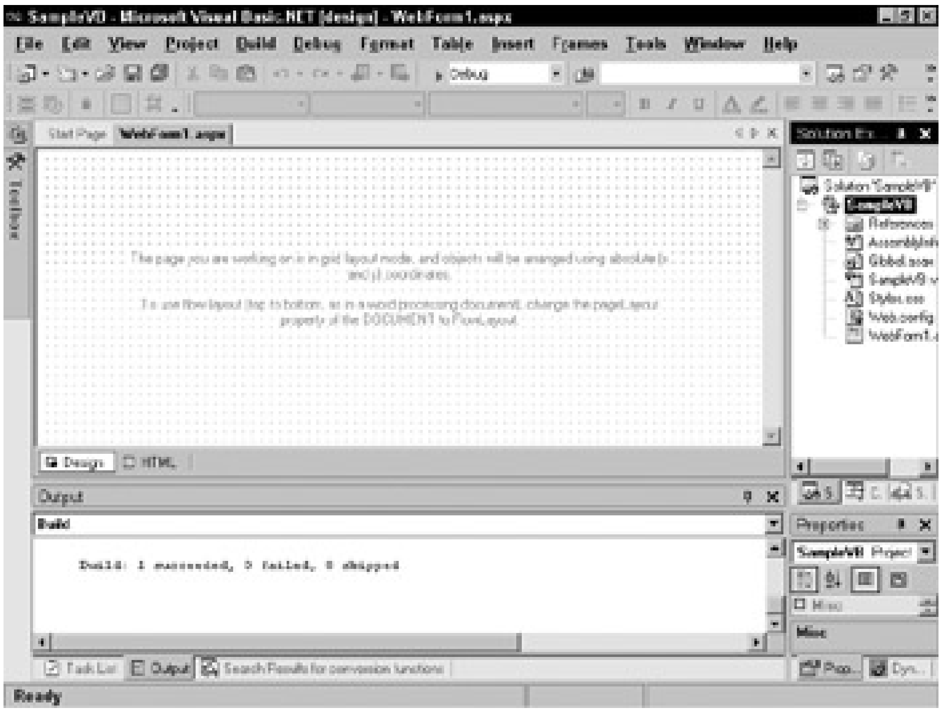
Location: http://NETSERVER-F90

Project will be created at http://NETSERVER-F90/SampleVB.

Figure 2-3: The New Project dialog box

4. Click OK to complete the process.

VS.NET displays the application, as shown in Figure 2-4. By default, the file WebForm1.aspx is selected and displayed. In addition to several other files, WebForm1.vb is also created. You can write the page logic in this file. This file is the *code behind file.*


SampleVB - Microsoft Visual Basic.NET [design] - WebForm1.aspx

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.

```
Build: 1 succeeded, 0 failed, 0 skipped
```

Figure 2-4: The VS.NET window with a new project

The WebForm1.aspx file is displayed in Design mode by default. To view the file in HTML mode, click HTML at the bottom of the WebForm1.aspx file window.

As you can see in HTML view, the language to be used on the page is VB. Any HTML text or code (in the `<% %>` block) within the `<Body> </Body>` block is rendered on the page when it is displayed in a Web browser.

The default background color of a page is white. You can change the background color of a page by setting the `bgcolor` attribute of the `<Body>` element. When you set this attribute, you are prompted to pick the color, as shown in Figure 2-5.

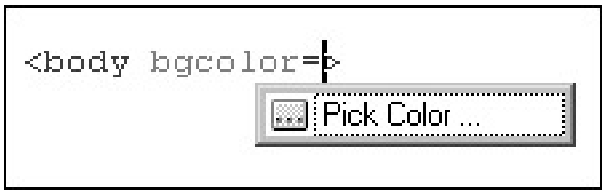


Figure 2-5: Setting the `bgcolor` attribute

When you select a color from the color palette, the corresponding color code is set as the value of the `bgcolor` attribute. A sample of such code is given as follows:

```
<Body bgcolor="#ccccff">
```

Write the following code within the `<Body> </Body>` element to display the text "Hello World":

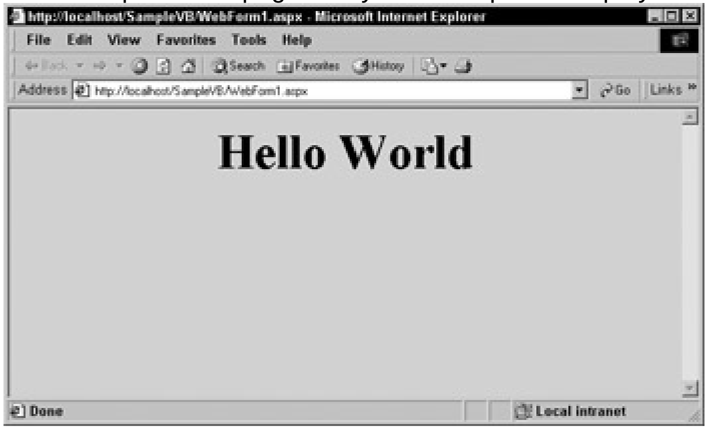
```
<% Response.Write(" <Font Size=10> <Center> <B> Hello World </B> </Center>
</Font>") %>
```

After you complete writing the code for your application, you need to build your application so that you can execute it on a Web server. To build the project, choose Build → Build.

Tip You can also build a project by pressing Ctrl + Shift + B.

When you build a project, the Web Form class file is compiled to a Dynamic Link Library (DLL) file along with other executable files in the project. The ASPX file is copied to the Web server without any compilation. You can change the ASPX file (only the visual elements of the page) without recompiling, because the ASPX file is not compiled. Later, when you run the page, the DLL and ASPX files are compiled into a new class file and then run.

The output of the page that you developed is displayed in Figure 2-6.



http://localhost/SampleVB/WebForm1.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/SampleVB/WebForm1.aspx
Go Links
Hello World
Done
Local intranet

Figure 2-6: A sample output of the Web page

Creating a C# Web Application

In addition to Visual Basic, you can also use C# to create ASP.NET Web applications. To do so, you need to create a Visual C# Web application project as follows:

1. Select File → New → Project. The New Project dialog box appears.
2. Select Visual C# Projects from the Project Types pane.
3. Select Web Application from the Templates pane. The Name box contains a default name of the application. The Location box contains the name of a Web server where the application will be created. However, you can change the default name and location. In this case, the name of the sample application is SampleCSharp.
4. Click OK to complete the process.

When you switch to HTML view of the WebForm1.aspx file, you'll notice that the language specified in the `@ Page` directive is C#. To create a Web page that displays

"Hello World," you simply need to write the following code in the `<Body> </Body>` block of the page:

```
<% Response.Write("<Font Size=10> <Center> <B> Hello World </B> </Center>
</Font>"); %>
```

Notice that the code in the <% %> block is terminated with a semicolon. This difference in syntax is due to the fact that the language for this page is C# and not VB.

When you build the application and execute it, a Web page appears in the browser displaying the text "Hello World."

Deploying an ASP.NET Web Application

After creating and testing your ASP.NET Web applications, the next step is deployment. *Deployment* is the process of distributing the finished applications (without the source code) to be installed on other computers.

In Visual Studio .NET, the deployment mechanism is the same irrespective of the programming language and tools used to create applications. In this section, you'll deploy the "Hello World" Web application that you created. You can deploy any of the application that was created by using VB or C#. Here, you'll deploy the application created by using VB. To do so, follow these steps:

1. Open the Web application project that you want to deploy. In this case, open the SampleVB project.
2. Select File → Add Project → New Project to open the Add New Project dialog box.
3. From the Project Types pane, select Setup and Deployment Projects. From the Templates pane, select Web Setup Project.
4. Change the default name of the project. In this case, change it to "SampleVBDeploy."
5. Click OK to complete the process. The project is added in the Solution Explorer window. Also, a File System editor window appears to the left, as shown in Figure 2-7. The editor window has two panes. The left pane displays different items. The right pane displays the content of the item selected in the left pane.

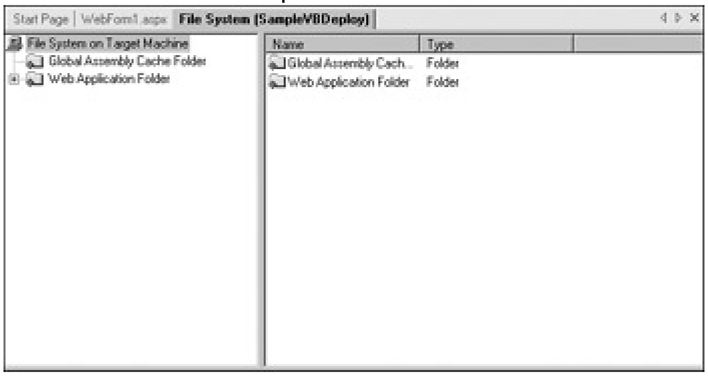


Figure 2-7: The Deployment editor

6. Select Web Application Folder in the left pane of the File System editor window. Then, from the Action menu, select Add → Project Output to open the Add Project Output Group dialog box, shown in Figure 2-8.

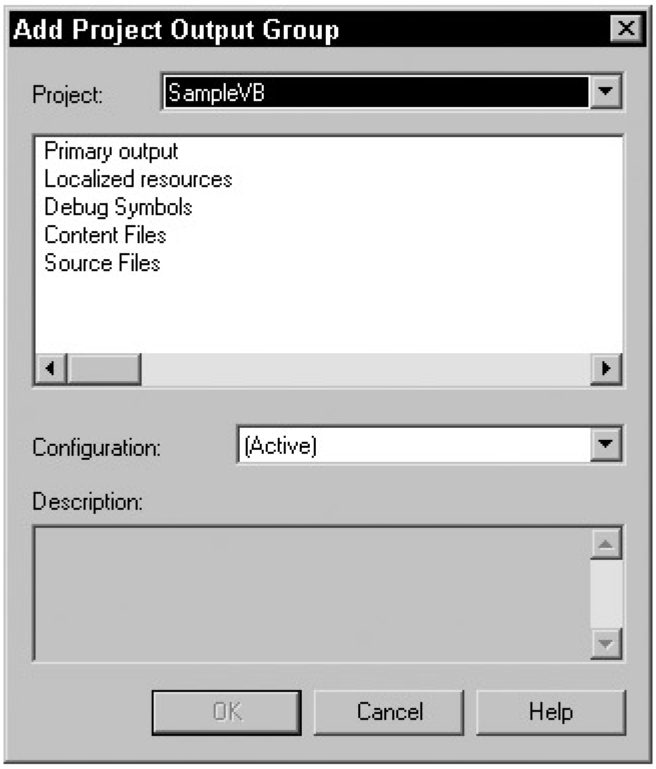


Figure 2-8: The Add Project Output Group dialog box

7. Verify that SampleVB is selected in the Project drop-down list. Then, select Primary Output and Content Files from the list.
8. Click OK. The output files and content files of the SampleVB project are added to the solution.
9. Select Web Application Folder in the File System editor and select Properties Window from the View menu to open the Properties window.
10. Set the `VirtualDirectory` property to a folder, <folder name>, that would be the virtual directory on the target computer where you want to install the application. By default, this property is set to `SampleVBDeploy`, which is the name of the Web Setup project that you added. In this case, set the property to `DeployedApplication`.

    Caution The <folder name> should be a new folder name and should not already exist on the target machine. Otherwise, the contents in the folder will be overwritten.

11. In the same Properties window of the Web Application Folder, set the `DefaultDocument` property to WebForm1.aspx. This property is used to set the default Web Forms page for the application.
12. Build the solution by selecting Build Solution from the Build menu.
13. After the solution is built successfully, a SampleVBDeploy.msi file is created in the Debug directory of the Web Setup project. The default path is \documents and settings\<*login name*>\My Documents\Visual Studio Projects\SampleVB\SampleVBDeploy\Debug\SampleVBDeploy.msi.
14. Copy the SampleVBDeploy.msi file to the Web server computer (c:\inetpub\wwwroot) where you want to deploy the application.
15. Double-click the SampleVBDeploy.msi file on the target computer to run the installer.

    Note To run the installer, you must have the install permissions on the computer. And, to install to the Web server, you must have IIS permissions.

After the installation is complete, you can run your application on the target computer. To do so, start Internet Explorer and enter `http://<computer`

`name>/DeployedApplication` in the address box. The "Hello World" page that you developed is displayed.

Summary

In this chapter, you learned how to set up the development environment for creating ASP.NET applications. Then, you learned how to create a simple ASP.NET application by using both VB and C#. Finally, you learned how to deploy an application on a Web server.

Chapter 3: Building Forms with Web Controls

Overview

The increased use of the Internet in the business scenario has shifted focus from desktop applications to Web-based applications. Because of this shift in focus, a Web development technology is needed that can combine the capabilities of different languages and simplify application development. Microsoft's response to this need is the release of ASP.NET, which provides a common Web development platform.

ASP.NET is a powerful programming platform that is used to develop and run Web-based applications and services. ASP.NET provides improved features, such as convenient debugging tools, over the earlier Web development technologies. ASP.NET provides a rich set of controls to design Web pages. Visual Studio .NET provides visual WYSIWYG (What You See Is What You Get) HTML editors. Therefore, you can design Web pages by simply dragging and dropping the controls. ASP.NET supports the C#, Visual Basic .NET, and JScript .NET languages, all of which you can use to build programming logic for your Web pages. You can choose which one of these languages to use based on your proficiency on a particular language. One of the most important features of ASP.NET is that it provides separate files for page presentation and programming logic, which simplifies Web application development. This chapter introduces you to designing simple Web pages by using basic Web controls. You'll also learn how to handle various events of these controls.

Introducing ASP.NET Web Forms

The ASP.NET Web Forms technology is used to create programmable Web pages that are dynamic, fast, and interactive. Web pages created using ASP.NET Web Forms are called *ASP.NET Web Forms pages* or simply *Web Forms pages*.

ASP.NET uses the .NET Framework and enables you to create Web pages that are browser independent. In addition to being browser independent, the following are some of the features that may lead you to select Web Forms over other technologies to create dynamic Web pages:

- Web Forms can be designed and programmed using Rapid Application Development (RAD) tools.
- Web Forms support a rich set of controls and are extensible, because they provide support for user-created and third-party controls.
- Any of the .NET Framework language can be used to program the ASP.NET Web Forms pages.
- ASP.NET uses the Common Language Runtime (CLR) of the .NET Framework and thus benefits from its features, such as type safety and inheritance.

Web Forms components
An ASP.NET Web Forms page consists of a user interface and programming logic. The user interface helps display information to users, while the programming logic handles user interaction with the Web Forms pages. The user interface consists of a file containing a markup language, such as HTML or XML, and server controls. This file is called a *page* and has .aspx as its extension.
The functionality to respond to user interactions with the Web Forms pages is implemented by using programming languages, such as Visual Basic .NET and C#. You can implement the programming logic in the ASPX file or in a separate file written in any CLR-supported language, such as Visual Basic .NET or C#. This separate file is called the *code behind* file and has either .aspx.cs or .aspx.vb as its extension depending on the language used. Thus, a Web Forms page consists of a page (ASPX file) and a code behind file (.aspx.cs file or .aspx.vb file).

Web Forms server controls
You can design a Web Forms page by using controls called *Web Forms server controls*. You can program the functionality to be provided for the server controls. The server controls are different from the usual Windows controls because they work within the ASP.NET Framework. The different types of server controls are described as follows:

- **HTML server controls:** These controls refer to the HTML elements that can be used in server code. The HTML elements can be converted into HTML server controls. To do so, you need to use attributes, such as `ID` and `RUNAT`, in the tags that are used to add the HTML controls. You can also add these controls to the page by using the HTML tab of the toolbox. The different tabs of the toolbox are shown in Figure 3-1.

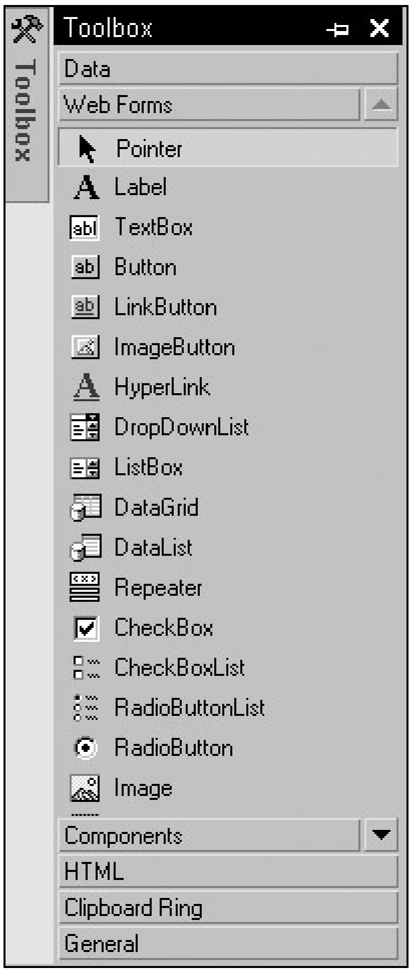


Figure 3-1: The Visual Studio .NET toolbox

- **ASP.NET server controls:** These controls do not map one-to-one to HTML server controls. ASP.NET server controls include traditional form controls, such as text boxes and buttons, and complex controls, such as tables.

- **Validation controls:** These controls are used to validate users' input. Validation controls can be attached to input controls to check the values entered.
- **User controls:** These controls are created from the existing Web Forms pages and can be used in other Web Forms pages.

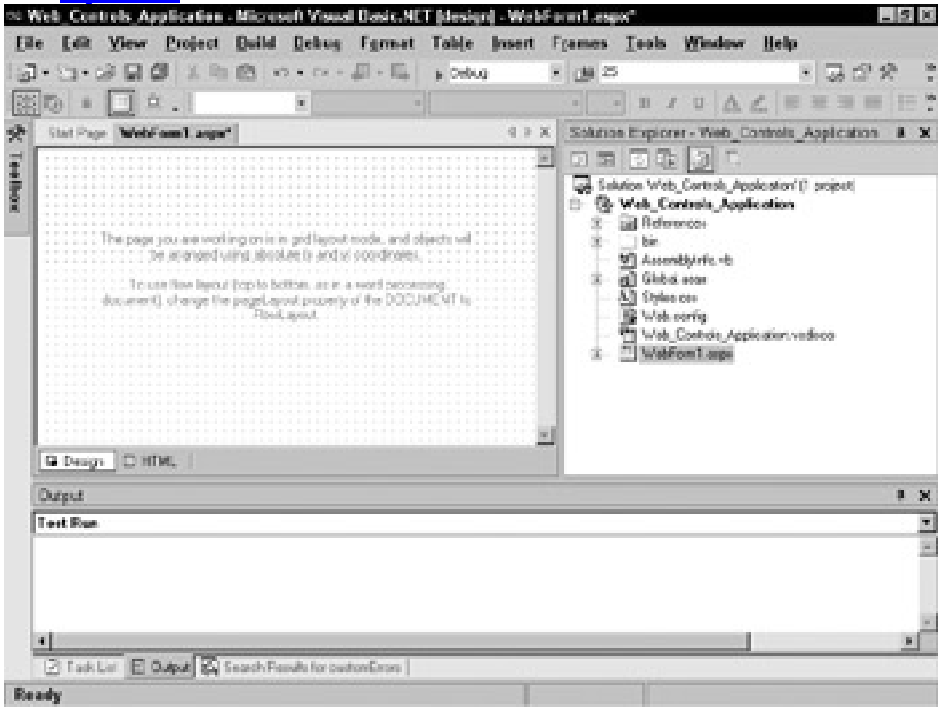
Creating Web Forms Application Projects

Before you use any server control to design a Web Forms page, you need to create an ASP.NET Web Application project. You can create either a Visual Basic .NET or a C# Web Application project, depending on the programming language you want to use. A Web Application project is always created on a Web server.

Note A Web server must be installed on the development computer to create a Web Application project.

The steps to create an ASP.NET Web Application project are as follows:

1. Select Start → Programs → Microsoft Visual Studio .NET 7.0 → Microsoft Visual Studio .NET 7.0 to start Visual Studio.NET.
2. Select File → New → Project to open the New Project dialog box.
3. Select Visual Basic Projects or Visual C# Projects in the Project Types pane.
4. Select ASP.NET Web Application in the Templates pane.
5. Specify the project name in the Name box, if necessary.
6. Specify the name of the computer where you want to create the application, in the Location box if necessary, and click OK. The name of the computer should be in the form http://*computer name*. A new Web Application project is displayed in the designer window, as shown in Figure 3-2.


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.
Web_Controls_Application - Microsoft Visual Basic.NET [design] - WebForm1.aspx*
File Edit View Project Build Debug Format Table Insert Frames Tools Window Help
Debug
Toolbox
Start Page
WebForm1.aspx*
Solution Explorer - Web_Controls_Application
Solution 'Web_Controls_Application' (1 project)
Web_Controls_Application
References
bin
AssemblyInfo.vb
Global.asax
Styles.css
Web.config
Web_Controls_Application.vsdisco
WebForm1.aspx
Design
HTML
Output
Test Run
Task List
Output
Search Results for customErrors
Ready

Figure 3-2: A Web Application project

Note By default, the Name and the Location boxes contain a project name and the computer name, respectively. However, you can change the default names.

When you create a Web Application project, the Application Wizard creates the necessary project files along with the page file and code behind class file as described:

- **WebForm1.aspx:** This page file consists of the user interface for the visual representation of the Web Forms page. The file has two views, Design and HTML. The default view is Design view.
    - **Design view:** This view represents the user interface for the Web Forms page. You can place controls directly from the toolbox to the Web Forms page in Design view. By default, the page layout of the Web Forms page is GridLayout. This layout enables you to accurately position controls on the page by using the absolute coordinates (X,Y) of the page. In addition to GridLayout, ASP.NET provides another page layout, which is called FlowLayout. In FlowLayout, you can add text to the page directly in Design mode. You can change the page layout from the default GridLayout to FlowLayout. To do so, right-click the page in Design view and select Properties from the context menu. Next, in the DOCUMENT Property Pages dialog box, from the Page Layout list box, select FlowLayout.
    - **HTML view:** This view represents the ASP.NET code for the Web Forms page. To open HTML view, click the HTML tab in the designer. When the Web Application project is a Visual Basic project or a C# project, the scripting language used in the HTML page is Visual Basic or C#, respectively.
- **WebForm1.aspx.cs or WebForm1.aspx.vb:** This file consists of the code to implement programming logic in the Web Forms page. You can view the code file by using the Show All Files icon in the Solution Explorer window. If the Web Application project is a Visual Basic project, you use Visual Basic .NET to implement the programming logic and the code file is called the *WebForm1.aspx.vb* file. Conversely, if the Web Application project is a C# project, you use C# to implement the programming logic and the code file is called the *WebForm1.aspx.cs* file. The code file (WebForm1.aspx.vb) appears within the WebForm1.aspx node as shown in Figure 3-3.

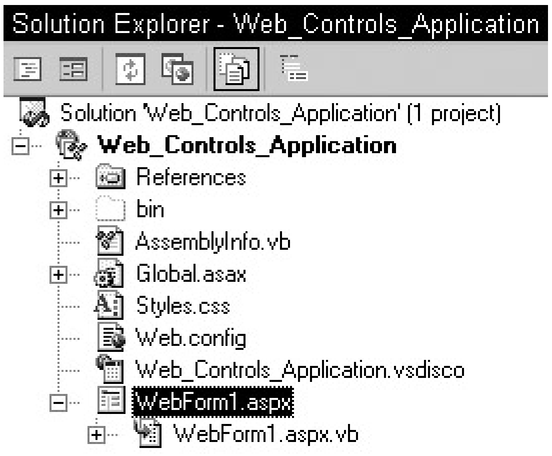

Solution Explorer - Web_Controls_Application
Solution 'Web_Controls_Application' (1 project)
Web_Controls_Application
References
bin
AssemblyInfo.vb
Global.asax
Styles.css
Web.config
Web_Controls_Application.vsdisco
WebForm1.aspx
WebForm1.aspx.vb

Figure 3-3: The Solution Explorer window showing all the files

Using Web Controls

You can add ASP.NET server controls to a Web Forms page by using either of the following two features:

- The Web Forms section of the toolbox
- The ASP.NET code

You add controls from the toolbox in Design view of the Web Forms page (the ASPX file). The toolbox categorizes the different types of controls in separate tabs, such as Web Forms, HTML, Components, and Data. You can use the HTML tab to add HTML controls and use the Web Forms tab to add the ASP.NET server controls to Web Forms. However, to make the HTML controls available for coding at the server end, these controls need to be converted to server controls. To do so, right-click the HTML control on the page and select Run As Server Control from the context menu. While selecting between HTML and Web server controls, using Web server controls is preferred, because they provide a rich object model and are adaptable to multiple browsers depending on browser capabilities. However, HTML server controls are preferred when migrating from the existing ASP pages to ASP.NET pages, because, unlike Web server controls, HTML server controls map directly to HTML tags.

You can also add a Web control to a page by using the ASP.NET code. You can access the ASP.NET code in the HTML view of the page (ASPX file). The actual syntax depends on the type of control that you want to add. The syntax used to add an HTML TextBox control is given as follows:

```
<input id="Text1" Type=text runat="server">
```

You can add ASP.NET server controls by using an Extensible Markup Language (XML) tag referenced as `asp`. When you add an ASP.NET TextBox control, the following syntax is generated for you:

```
<asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
```

Note When you use the toolbox to add Web controls in Design view, the corresponding ASP.NET syntax is automatically generated.

In the preceding code, the XML tag `asp` maps to the System.Web.UI..WebControls namespace. This is different from the HTML server controls where the `input` tag lacks any such mapping. However, the Web server controls use the `runat=server` attribute, which is similar to the HTML server controls.

You can also programmatically add a control at run time. The following VB.NET code snippet demonstrates how to add a TextBox control at run time:

```
Dim TextBox1 as New TextBox()

Controls.Add(TextBox1)
```

Every control has specific properties and methods. You can set control properties to modify the appearance or behavior of controls. For example, you can set the font, color, and size of a control. You can use the control methods to perform a specific task, such as moving a control. You can set control properties at design times by using the Properties window or at run time by using the code. Every control has a property called `ID` that is used for the unique identification of the control. You can set the property of a control at run time by using the following syntax:

```
ControlID.PropertyName=Value
```

In this syntax:

- `ControlID` represents the `ID` property of the control.
- PropertyName represents the control property.
- Value represents the value assigned to PropertyName, which is a control's property.

Figure 3-4 displays a Web Forms page that contains almost every basic Web control. You can see that the Web Forms page is a user registration form. The form is designed to accept user input through various controls. After filling out the form, a user may click the Register button to complete the registration process. Alternatively, a user may click the Reset button to clear the values entered in the form. Table 3-1 lists the IDs of the different controls used in the form. The section that follows describes some of the basic Web controls in detail.

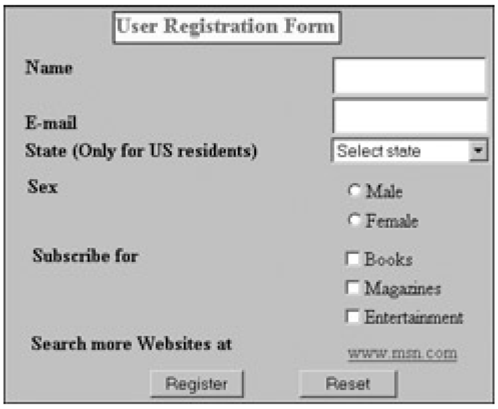


Figure 3-4: A Web Forms page

Table 3-1: IDs of different controls

Control	Contains	ID
TextBox	Name	UserName
TextBox	E-mail	Email
DropDownList	State	USStateList
RadioButtonList	Sex	SexOption
CheckBoxList	Subscriptions	SubscriptionOption
HyperLink	Search	SearchLink
Button	Register	RegisterButton
Button	Reset	ResetButton

Label control

You use the Label control to display static text in a Web Forms page that users cannot edit. When you add a Label control, the text "Label" appears as its caption. However, you can use the `Text` property to modify the caption. Table 3-2 lists some of the properties of the Label control.

Table 3-2: Properties of the Label control

Property	Description
`Text`	Represents the caption of a label
`BackColor`	Represents the background color of a label
`ForeColor`	Represents the font color of a label
`Visible`	Indicates whether or not a label is visible

You can also change the text of a label by using the following code:

```
Label1.Text="Welcome"
```

In this code, `Label1` is the ID of the Label control for which you want to change the state. You can use the `Visible` property of the Label control to make it visible or not. For example, in the following code, the `Visible` property is set to False, making the label invisible:

```
Label1.Visible=False
```

TextBox control

You use the TextBox control to get information, such as text, numbers, and dates, from users in a Web Forms page. You can set the `TextMode` property of a TextBox control to set the type as a single-line, password, or multiline TextBox control. By default, a TextBox control is a single-line text box that allows users to type characters in a single line only. A password TextBox control is similar to the single-line text box, but masks the characters that are typed by users and displays them as asterisks (*). A multiline TextBox control allows users to type multiple lines and wrap text.

The appearance of the TextBox control can be modified by using properties such as `BackColor` or `ForeColor`. Table 3-3 lists some of the properties of the TextBox control.

Table 3-3: Properties of the TextBox control

Property	Description
`Text`	Represents the text to be displayed in the TextBox control. Also, you can use this property to set or retrieve the text to or from a TextBox control at run time.
`MaxLength`	Represents the number of characters that a user can type in the TextBox control.
`Width`	Represents the width of a TextBox control. This property takes value in pixels.
`Columns`	Represents the width of a TextBox control in characters. If you set both the `Width` and `Columns` properties, the `Width` property takes precedence over the `Columns` property. The default value is 0.
`TextMode`	Represents the behavior of the TextBox control, such as single-line, multiline, or

Table 3-3: Properties of the TextBox control

Property	Description
	password. By default, the control is a single-line text box. To set a password text box, set this property to TextBoxMode.Password. To set a multiline text box, set this property to TextBoxMode.MultiLine. The values SingleLine, MultiLine, and Password are part of an enum called TextBoxMode. You cannot specify it directly.
`Height`	Represents the vertical size of the TextBox control and takes value in pixels.
`Rows`	Represents the vertical size of the MultiLineTextBox control and takes value in number of rows The default value is 0.
`Wrap`	Represents the word wrap behavior in a multiline TextBox control. The text wraps automatically if the value is set to True. However, a user must press a carriage return to move to a next line if the value is set to False. The default value is True.

Note The `Height` and `Width` properties do not work in browsers that do not support Cascading Style Sheets (CSS). The CSS is a list of CSS styles that is used to apply a general rule to attributes of a set of elements.

CheckBox and CheckBoxList controls

Check boxes provide you with independent choices or options that you can select. You can add check boxes to a Web Forms page by using either the CheckBox control or the CheckBoxList control. The CheckBox control is a single check box that you can work with. On the other hand, the CheckBoxList control is a collection of several check boxes. After you add the CheckBoxList control, you need to add a list of items to it. To do so:

1. Display the Properties window of the CheckBoxList control.

2. Click the ellipsis button for the Items property of the CheckBoxList control.
3. In the ListItem Collection Editor dialog box, click Add to create a new item. A new item is created and its properties are displayed in the Properties pane of the dialog box.
4. Verify that the item is selected in the Members list, and then set the item properties. Each item is a separate object and has following properties:
   - `Text`: Represents the text to be displayed for the item in the list.
   - `Value`: Represents the value associated with the item without displaying it. For example, you can set the `Text` property of an item as the city name and the `Value` property to the postal code of the city. Thus, you can keep the `Text` and `Value` properties different when you do not want the actual value to be displayed to the user.
   - `Selected`: A Boolean value that indicates whether or not the item is selected.

In addition to adding the CheckBoxList control and the member items at design time, you can programmatically add them at run time. To do so, you use the following VB.NET code:

```
Dim CheckBoxList1 As New CheckBoxList()

Controls.Add(CheckBoxList1)

CheckBoxList1.Items.Add("Check1")
```

The `Add()` method of the Items class can take either a string argument or a ListItem object. This code snippet uses the `Add()` method that takes one string argument to represent the text of the item.
The `Add()` method can also take a ListItem object as an argument. The ListItem constructor can take one argument (one string to represent the text of the item) or two arguments (one string for the text and another string for the value of the item). The following code snippet explains the usage of the ListItem object in the `Add()` method:

```
Dim ListItem1 as New ListItem("Check1","check")

CheckBoxList1.Items.Add(ListItem1)
```

The following VB.NET code snippet assumes that the Web Forms page contains a TextBox control and a Button control. The following code is also associated with the `Click` event of the button. When the user enters a number in the text box and clicks the button, the specified number of check boxes is added to the CheckBoxList control:

```
'Create a CheckBoxList object
Dim CheckBoxList1 as New CheckBoxList()
'Adding the CheckBoxList control to the page
Controls.Add(CheckBoxList1)


'Declare the total number of items
Dim ChkCount as Integer


'Declare the current number of items
Dim ChkCtr as Integer


'Accept the total number of items
ChkCount = Val(TextBox1.Text)
```

```
For ChkCtr = 0 To ChkCount-1
    CheckBoxList1.Items.Add("Check" & ChkCtr)
Next ChkCtr
```

The choice between using the CheckBox control and the CheckBoxList control depends on application needs. The CheckBox control provides more control over the layout of the check boxes on the page. For instance, you can set the font and color of the check boxes individually or include text between different check boxes. On the other hand, the CheckBoxList control is a better choice if you need to add a series of connected check boxes, such as check boxes to represent areas of interest.
You can identify whether a check box is checked or not by using the `Checked` property of the CheckBox control. The `Checked` property returns a Boolean value, as indicated in the code that follows. If the control is checked, it returns True; otherwise, it returns False.

```
Dim CheckBox1 as New CheckBox()
Dim IsChk as Boolean
IsChk=CheckBox1.Checked
```

If you have a CheckBoxList control and you want to identify the item that has been checked, you use either the `SelectedIndex` or `SelectedItem` property of the control. The `SelectedIndex` property returns an integer value indicating the index (the first item has an index 0) of the selected item. This property returns -1 if nothing is selected.

```
Dim i As Integer
i=CheckBoxList1.SelectedIndex()
```

The `SelectedItem` property, on the other hand, returns the selected item:

```
Dim ListItem1 As New ListItem()
ListItem1=CheckBoxList1.SelectedItem()
```

You can also access the `Text`, `Value`, or `Selected` property of the item. For example, the following code retrieves the `Text` property of the selected item:

```
Dim CityName as String
CityName=CheckBoxList1.SelectedItem.Text
```

When you add a CheckBox control to a page, you can set the caption of the check box by setting the `Text` property. Then, you can change the orientation of the caption by setting the `TextAlign` property. Table 3-4 describes some of the properties of the CheckBox and CheckBoxList controls.

Table 3-4: Properties of the CheckBox and CheckBoxList controls

Property	Available with	Description
`Text`	CheckBox	Represents the caption of the CheckBox control.
`TextAlign`	CheckBox and CheckBoxList	Represents the text orientation of the CheckBox and CheckBoxList controls.
`Items`	CheckBoxList	Represents the

Table 3-4: Properties of the CheckBox and CheckBoxList controls

Property	Available with	Description
		collection of individual check boxes in the CheckBoxList control. Each item has three properties, `Text`, `Value`, and `Selected`, associated with it.

RadioButton and RadioButtonList controls

Radio buttons provide a set of choices or options that you can select. You can add radio buttons to a Web Forms page by using either the RadioButton control or the RadioButtonList control. The RadioButton control is a single radio button. On the other hand, the RadioButtonList control is a collection of radio buttons. Radio buttons are seldom used singly. Usually, you use radio buttons in a group. A group of radio buttons provides a set of mutually exclusive options — you can select only one radio button in a group. You can group a set of radio buttons in two ways:

- Place a set of RadioButton controls on the page and assign them manually to a group. To do so, you can use the `GroupName` property.
- Place a RadioButtonList control on the page; the radio buttons in the control are automatically grouped.

After you add a RadioButtonList control, you need to add the individual radio buttons. You can do so by using the `Items` property in the same way as you do for the CheckBoxList control.

You add the items to a RadioButtonList control at run time in the same way as you add items to a CheckBoxList control. The following VB.NET code snippet demonstrates how to add items to a RadioButtonList control programmatically:

```
Dim RadioButtonList1 As New RadioButtonList()

Controls.Add(RadioButtonList1)

RadioButtonList1.Items.Add("Radio1")
```

You can use the `Checked` property of the RadioButton control to identify whether or not the control is selected. For the RadioButtonList control, you can access the index of the selected item by using the `SelectedIndex` property and access the selected item by using the `SelectedItem` property of the control.

Table 3-5 describes some of the properties of the RadioButton and RadioButtonList controls. Like the CheckBox control, the RadioButton control offers more control over the layout of the radio buttons on the page.

Table 3-5: Properties of the RadioButton and RadioButtonList controls

Property	Available with	Description
`Text`	RadioButton	Represents the caption of the RadioButton control.

Table 3-5: Properties of the RadioButton and RadioButtonList controls

Property	Available with	Description
`TextAlign`	RadioButton and RadioButtonList	Represents the text orientation of the RadioButton and RadioButton List controls.
`Items`	RadioButtonList	Represents the collection of the individual radio buttons in the RadioButton List control. Each item has three properties associated with it: `Text`, `Value`, and `Selected`.

ListBox control

The ListBox control is a list of predefined items and allows users to select one or more items from the list. The ListBox control is a collection of items. The individual list items can be added by using the `Items` property of the ListBox control.

You can add list items to the ListBox control in the same way you add items to the CheckBoxList and RadioButtonList controls. You can access the index of the selected item by using the `SelectedIndex` property and access the selected item in the list by using the `SelectedItem` property of the control.

Table 3-6 describes some of the properties of the ListBox control.

Table 3-6: Properties of the ListBox control

Property	Description
`Items`	Represents the collection of list items in the ListBox control. Each list item has three properties associated with it: `Text`, `Value`, and `Selected`.
`Width`	Represents the size of a ListBox control and takes value in pixels.
`Height`	Represents the vertical size of the ListBox control and takes value in pixels.

Table 3-6: Properties of the ListBox control

Property	Description
`Rows`	Represents the vertical size of the ListBox control and takes value in number of rows. If the control contains more than the specified number of items, the control displays a vertical scroll bar.
`SelectionMode`	Represents the number of items that can be selected. To allow users to select only one item, set the `SelectionMode` property to ListSelectionMode.Single. To allow users to select multiple items, set the `SelectionMode` property to ListSelectionMode. Multiple. ListSelectionMode is the enum that allows you to specify the selection mode. To select more than one item, users can hold the Ctrl or Shift key while clicking multiple items. This is possible only when you set the `SelectionMode` property to ListSelectionMode.Multiple.

DropDownList control

The DropDownList control allows users to select an item from a set of predefined items — each item is a separate object with its own properties, such as `Text`, `Value`, and `Selected`. You can add these predefined items to a DropDownList control by using its `Items` property. Unlike the ListBox control, you can select only one item at a time, and the list of items remains hidden until a user clicks the drop-down button.

You can add list items to the DropDownList control in the same way you add items to the CheckBoxList, RadioButtonList, and ListBox controls. You can access the index of the selected item by using the `SelectedIndex` property and access the selected item in the list by using the `SelectedItem` property of the control.

Table 3-7 describes some of the properties of the DropDownList control.

Table 3-7: Properties of the DropDownList control

Property	Description
`Items`	Represents the

Table 3-7: Properties of the DropDownList control

Property	Description
	collection of items in the DropDownList control. Each item has three properties associated with it: `Text`, `Value`, and `Selected`.
`Width`	Represents the width of a DropDownList control and takes value in pixels.
`Height`	Represents the vertical size of the DropDownList control and takes value in pixels.

HyperLink control
The HyperLink control creates links on a Web page and allows users to navigate from one page to another in an application or an absolute URL. You can use text or an image to act as a link in a HyperLink control. When users click the control, the target page opens. Table 3-8 describes some of the properties of the Hyperlink control.

Table 3-8: Properties of the HyperLink control

Property	Description
`Text`	Represents the text displayed as a link.
`ImageUrl`	Represents the image displayed as a link. The image file should be stored in the same application project.
`NavigateUrl`	Represents the URL of

Table 3-8: Properties of the HyperLink control

Property	Description
	the target page.

Note The `ImageUrl` property takes precedence when both the `Text` and the `ImageUrl` properties are set.

The following code illustrates how to set the `NavigateUrl` property programmatically:

```
Dim HyperLink1 as New HyperLink()
HyperLink1.NavigateUrl="http://www.msn.com"
```

Table control

A table is used to display information in a tabular format. A table consists of rows and columns. The intersection of a row and a column is called a cell. You can add a table to a Web Forms page by using the Table control. This control displays information statically by setting the rows and columns at design time. Also, you can program the Table control to display information dynamically at run time.

You can add rows at design time by setting the `Rows` property, which represents a collection of TableRow objects; a TableRow object represents a row in the table. You can add cells to a table row by setting the `Cells` property of the TableRow object. The `Cells` property represents a collection of TableCell objects; a TableCell object represents a cell in a table row. Thus, to set rows and columns of a table at the design time, you first add the Table control to the form. Then, set the `Rows` property of the Table control to add TableRow objects. Finally, set the `Cells` property of the TableRow objects to add TableCells objects. The steps are given as follows:

1. Display the Properties window of the Table control.
2. Click the ellipsis button for the Rows property of the Table control.
3. In the TableRow Collection Editor dialog box, click Add to create a new row. A new row is created and its properties are displayed in the Properties pane of the dialog box.
4. Verify that the row is selected in the Members list, and then click the ellipsis button for the Cells property to add a cell for the row.
5. In the TableCell Collection Editor dialog box, click Add to create a new cell. A new cell is created and its properties are displayed at the right side of the dialog box. Table 3-9 describes some of the properties of the TableCell object.

You can also add the rows and columns (cells) to a table at run time programmatically. To do so, you first need to create the TableRow and TableCell objects:

```
Dim Table1 as New Table()
Dim TableRowObj As New TableRow()
Dim TableCellObj As New TableCell()
```

Then, you need to add the TableCell object to the TableRow object:

```
TableRowObj.Cells.Add(TableCellObj)
```

Finally, you need to add the TableRow object to the Table control. If the ID of the Table control is Table1, use the following code to add the TableRow object to the Table control:

```
Table1.Rows.Add(TableRowObj)
```

Table 3-9: Properties of the TableCell object

Property	Description
`ColumnSpan`	Represents the number

Table 3-9: Properties of the TableCell object

Property	Description
	of columns that the cell spans. By default, this property is set to 0.
`RowSpan`	Represents the number of rows that the cell spans. By default, this property is set to 0.
`VerticalAlign`	Represents the vertical alignment, such as top and bottom of the cell.
`HorizontalAlign`	Represents the horizontal alignment, such as left and right of the cell.
`Text`	Represents the text contents of a cell.

The following Visual Basic .NET code snippet demonstrates how to add rows and cells (columns) at run time. Assume that the Web Forms page contains a Table control, a Button control, and two TextBox controls (to accept the number of rows and cells that need to be added to the table). The following code is also associated with the `Click` event of the Button control:

```
' Declare the total number of rows
Dim RowCnt As Integer
'Declare the current row counter
Dim RowCtr As Integer

'Declare the total number of cells
  Dim CellCtr As Integer
'Declare the current cell counter
Dim CellCnt As Integer

'Accept the total number of rows and columns from the user
RowCnt = Val(TextBox1.Text)
```

```
CellCnt = Val(TextBox2.Text)

For RowCtr = 1 To RowCnt
   'Creating a TableRow object
    Dim TableRowObj As New TableRow()
      For CellCtr = 1 To CellCnt
        'Creating a TableCell object
         Dim TableCellObj As New TableCell()
         TableCellObj.Text = RowCtr & "Row, " & CellCtr & " Cell "
         'Add the new TableCell object to row
         TableRowObj.Cells.Add(TableCellObj)
      Next
      'Add new row to table
       Table1.Rows.Add(TableRowObj)
   Next
```

Image control

The Image control allows users to display images in a Web Forms page and manage them at design time or at run time. After you add an Image control to a Web Forms page, you need to set the image to be displayed in the control. You can do so by using the `ImageUrl` property. Table 3-10 describes some of the properties of the Image control.

Table 3-10: Properties of the Image control

Property	Description
`ImageUrl`	Represents the URL of the image to be displayed in the control.
`ImageAlign`	Represents the alignment of the image with respect to the other controls in the page and not just the text.
`AlternateText`	Represents the text that is displayed as a tooltip or when the image cannot be loaded.

Consider the following code that is used to set the `ImageUrl` property of the Image control in the `Page_Load` event:

```
Dim Img1 as New Image()
```

Img1.ImageUrl="Rose.gif"

Button, LinkButton, and ImageButton controls

The Button control on a Web Forms page is used to perform an event, such as form submit, on the server. You can create three types of server control buttons:

- **Button:** Represents a standard button.
- **LinkButton:** Represents a button that can act as a hyperlink in a page. However, a LinkButton control causes the page to be submitted to the server.
- **ImageButton:** Represents a graphical button to provide a rich button appearance. You can set the ImageUrl property to point to a specific image.

Table 3-11 describes some of the properties of the server control buttons.

Table 3-11: Properties of the button server control

Property	Available with	Description
`Text`	Button and LinkButton	Represents the text to be displayed on the Button and the LinkButton controls.
`Enabled`	Button, LinkButton, and ImageButton	Represents whether or not the button is available at run time. By default, this property is set to True, indicating that the button is available at run time.
`ImageUrl`	ImageButton	Represents the URL of the image to be displayed in the control.
`AlternateText`	ImageButton	Represents the text that is displayed as a tooltip or when the image cannot be loaded.

Working with Events

A Web Forms application provides fast, dynamic, and user-interactive Web Forms pages. When users interact with different Web controls on a page, events are raised. In the traditional client forms or client-based Web applications, the events are raised and handled on the client side. However, in Web Forms applications, the events are raised either on the client or on the server, but are always handled on the server. ASP.NET server controls support only server-side events, while HTML server controls support both server-side and client-side events.

Round trips

Most Web pages require processing on the server. For example, consider an Orders Web page used to receive orders on the Web. When a user enters a value for the quantity of a product to be bought, the page must check on the server to see whether or not the quantity requested is available. This type of dynamic functionality is accomplished by handling server control events. Whenever a user interaction requires some kind of processing on the server, the Web Forms page is submitted to the server, processed, and then returned to the browser (client). This sequence is called a *round trip.* Figure 3-5 describes round trips.

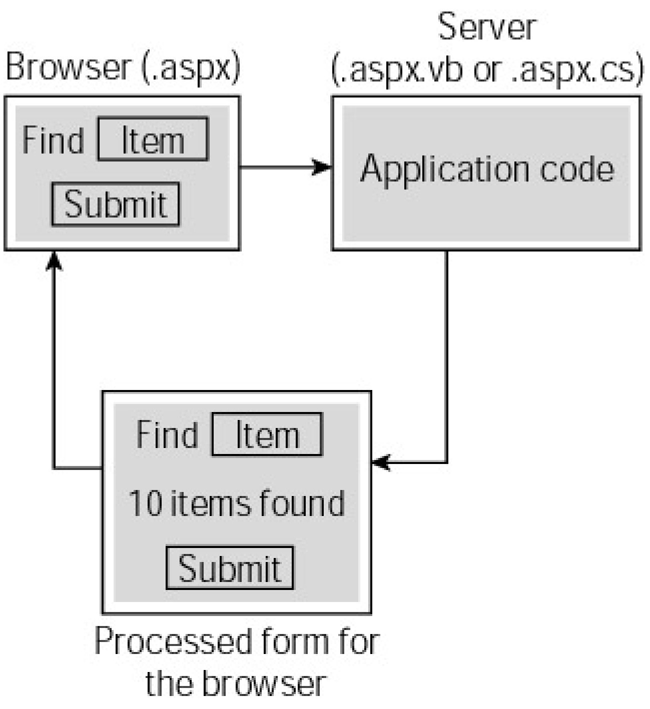


Figure 3-5: A round trip

Most of the user interactions with the server controls result in round trips. Because a round trip involves sending the Web Forms page to the server and then displaying the processed form in the browser, the server control events affect the response time in the form. Therefore the number of events available in Web Forms server controls is limited, usually to `Click` events. The events that occur quite often, such as the `OnMouseOver` event, are not supported by server controls. However, some server controls support events that occur when the control's value changes. Table 3-12 describes the events associated with different ASP.NET server controls.

Table 3-12: Events associated with ASP.NET server controls

Control(s)	Event	Description
TextBox	`TextChanged`	Occurs when the content of the text box is changed.
RadioButton and CheckBox	`CheckedChanged Checked`	Occurs when the value of the property changes.
RadioButtonList, CheckBoxList,ListBox, andDropDownList	`SelectedIndexChanged`	Occurs when you change the selection in the list.
Button, LinkButton, and ImageButton	`Click`	Occurs when you click the button. This event causes the form to be

Table 3-12: Events associated with ASP.NET server controls

Control(s)	Event	Description
		submitted to the server.

By default, only the `Click` event of the Button, LinkButton, and ImageButton server controls causes the form to be submitted to the server for processing — the form is said to be *posted back* to the server. The `Change` events associated with other controls are captured and cached and do not cause the form to be submitted immediately. When the form is posted back (as a result of a button click), all the pending events are raised and processed. No particular sequence exists for processing these `Change` events, such as `TextChanged` and `CheckChanged` on the server. The `Click` event is processed only after all the `Change` events are processed.
You can set the change events of server controls to result in the form post back to the server. To do so, modify the `AutoPostBack` property to True.

Event handlers
When the events are raised, you need to handle them for processing. The procedures that are executed when an event occurs are called *event handlers*. An event handler is associated with the corresponding event by using the `WithEvents` and `Handles` keywords. The `WithEvents` keyword is used to declare the control generating an event. For example, when you declare a control, say `Image1` as 'Protected WithEvents Image1 As System.Web.UI.WebControls.Image', the `WithEvents` keyword specifies that `Image1` is an object variable used to respond to events raised by the instance assigned to the variable. The `Handles` keyword is used to associate the event handler with the event, which is raised by the control. The control in turn is declared by using the `WithEvents` keywords.

Event handlers are automatically created when you double-click the control in Design mode of the form. For example, the following code is generated when you double-click a Button control whose ID is RegisterButton. You can then write the code in the event handler to perform the intended task.

```
Public Sub RegisterButton_Click(ByVal sender As System.Object,

ByVal e As System.EventArgs) Handles RegisterButton.Click


End Sub
```

In this code:

- The procedure `RegisterButton_Click` is the event handler for the `Click` event of the button with ID RegisterButton. The `Handles` keyword associates the event with the event handler.
- The procedure takes two arguments. The first argument contains the event sender. An *event sender* is an object, such as a form or a control, that can generate events. The second argument contains the event data.

Implementing the events and event handlers
After discussing the events and event handlers in detail, we'll now implement them for the Web Forms page shown in Figure 3-3, earlier in the chapter.
In Figure 3-3, when you click the Register button, another page should open displaying a relevant message along with the username entered in the UserName TextBox control. Before you can proceed to write the event handlers, you need to add another Web Forms page (the target page) to the Application project. To do so:

1. Select Project → Add Web Form. The Add New Item dialog box opens.
2. Specify the name of the Web Forms page and click Open. In this case, leave the default name of the Web Forms page, WebForm2.aspx.

Tip When you add a Web Forms page to a project, the name of the Web Forms page automatically takes the next number. For example, if WebForm1 already exists in the project, the default name of the new Web Forms page would be WebForm2.

Because the target page (WebForm2) should display a message, you need to add a Label control to this page. Set the `ID` property of this Label control to `MessageLabel`. To implement this functionality, you need to write the following code in the `Click` event of the Register button (in the WebForm1 page):

```
Response.Redirect("WebForm2.aspx?strTextValue=" & "Hi," &

UserName.Text & ", You have been successfully registered")
```

In this code, the `Response.Redirect` method takes the URL of the target page. The URL specifies another form named WebForm2.aspx (that you added) and passes a text string along with the value in the TextBox control whose ID is UserName in a variable called `strTextValue`.

After passing the text in the `strTextValue` variable, the Label control in the target form, WebForm2, must be initialized in the `Init` procedure of the form as follows:

```
MessageLabel.Text = Request.QueryString("strTextValue")
```

In this code, the value stored in the `strTextValue` is set as the caption of the label with ID `MessageLabel` in WebForm2.

The Web Forms page displayed in Figure 3-3 also contains a Reset button. When you click the Reset button, all the controls should be empty. To implement this functionality, use the following code:

```
UserName.Text = ""

Email.Text = ""

USStateList.ClearSelection()

SexOption.ClearSelection()

SubscriptionOptions.ClearSelection()
```

In this code:

- The `Text` property of the TextBox controls with IDs UserName and Email are set to a null value.
- ClearSelection is a method of the list controls, such as ListBox, DropDownList, CheckBoxList, and RadioButtonList controls. The method is used to clear any selection made in the list.

Handling post back

As mentioned earlier, the Web Forms page is posted back to the server only when a Button, LinkButton, or ImageButton ASP.NET server control is clicked. After the page is posted to the server, it is processed there. You can respond to a button event in one of the following ways:

- Write an event handler for the `Click` event of the button.
- Write the event handler for the Load event of the Web Forms page. The Load event is generated when the form is loaded from the server to the client (browser). You can use the IsPostBack property in the Load event to determine whether the page has been processed for the first time or by a button click. To understand the IsPostBack property better, consider the following code in the Page_Load event of the WebForm1 page. The following code checks whether the IsPostBack property is True. If it is, the Visible property of the Register button is set to False.

- Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
- System.EventArgs) Handles MyBase.Load

- If ResetButton.Page.IsPostBack = True Then
- RegisterButton.Visible = False
- End If

End Sub

Using the view state
In all Web applications, whenever a Web page is processed at the server, the page is created from scratch. In traditional Web applications, the server discards the page information after processing and sending the page to the browser. Because the page information is not preserved on the server, the Web pages are called *stateless*. However, the Web Forms framework works around this limitation and can save the state information of the form and its controls. To implement this, the Web Forms framework provides the following options:

- **The ViewState:** The framework automatically saves the state of the page and its current properties, and the state of the server controls and their base properties, with each round trip.
- **The State Bags:** Every page has a state bag that stores values to be restored in the next round trip.

The framework automatically stores and restores page information with each round trip. So, you do not need to worry about storing and restoring the page information with each round trip.

The ViewState contains the state of all the controls on a page between requests sent to the server. The state information is stored as hidden form fields as name-value pairs in the System.Web.UI.StateBag object. When you view an ASP.NET page in a browser, you can see the ViewState for this page by displaying the source code of the page. To do so, select View → Source in the browser in which the ASP.NET page is displayed. The ViewState thus is stored in a page rather than in the server. For complex pages that contain many controls, the ViewState information is too large to be stored in a page and might affect the performance of the page. This is the only disadvantage with ViewState. By default, the ViewState is enabled for all the server controls. All the server controls have the `EnableViewState` property set to True by default. Therefore, to take advantage of the ViewState, you do not need to do anything explicitly. However, as already mentioned, due to performance issues, you can set the `EnableViewState` property to False to disable the ViewState. If you do not want to maintain the state of any of the server controls on an ASP.NET page, you can set the `EnableViewState` property of the page to False:

```
<%@ Page EnableViewState="false" %>
```

Summary

This chapter served as a foundation for creating Web Forms applications. This chapter introduced you to the basic Web controls used for designing Web Forms pages. You learned the basic steps to create a Web Application project. Then, you learned the usage and properties of Web controls. The chapter also introduced you to events. You learned how to handle server-side events. Finally, you learned to handle post back and use the view state.

Chapter 4: Using Rich Web Controls

Overview

ASP.NET has brought about a complete change in the way controls are used in Web applications. In addition to the client-side rendering of controls, ASP.NET provides controls that can be rendered on the server side. This allows server-side processing, and thus provides dynamic Web pages resulting in a rich and improved user experience. The previous chapter discussed the basic Web controls. In addition to these Web controls, there are specific Web controls that have more complex and rich functionality. These controls are called Rich Web controls, examples of which are the AdRotator and Calendar controls. Some of the Rich Web controls include:

- TreeView
- TabStrip
- MultiPage
- Toolbar

In this chapter, you will learn about the functionality of these Rich Web controls and learn how to work with them in ASP.NET.

Using the AdRotator Control

The AdRotator control is used to display flashing ads, such as banner ads and news flashes on Web pages. The control is capable of displaying ads randomly, because the control refreshes the display every time the Web page is refreshed, thereby displaying different ads for different users. Also, you can assign priorities to the ads so that certain ads are displayed more frequently than others.

You can add the AdRotator control in an ASP.NET Web page by using the following syntax:

```
<asp:AdRotator
propertyname = propertyvalue
propertyname = propertyvalue
>
</asp:AdRotator>
```

Alternatively, you can use the toolbox provided with VS.NET to add the control to the page. When you do so, the code is automatically generated and can be seen in the HTML view of the ASPX file.

Properties of the AdRotator control

Along with the properties that are inherited from the System.Web.UI.Control base class, the AdRotator control has three additional properties:

- `AdvertisementFile`
- `KeywordFilter`
- `Target`

This section describes these properties in detail.

AdvertisementFile

The AdvertisementFile property represents the path to an Advertisement file. The *Advertisement file* is a well-formed XML document that contains information about the image to be displayed for advertisement and the page to which a user is redirected when the user clicks the banner or image. The following is the syntax of the Advertisement file:

```
<Advertisements>
  <Ad>
    <ImageUrl>
```

```
      URL of the image to display
    </ImageUrl>
    <NavigateUrl>
      URL of the page to navigate to
    </NavigateUrl>
    <AlternateText>
      Text to be displayed as ToolTip
    </AlternateText>
    <Keyword>
      keyword used to filter
    </Keyword>
    <Impressions>
      relative weighting of ad
    </Impressions>
  </Ad>
</Advertisements>
```

Note The Advertisement file must be a well-formed XML document, as the AdvertisementFile property of the AdRotator control needs to be set to an XML file.

The following are the different elements used in the Advertisement file:

- `ImageUrl`: Specifies an absolute or relative URL to an image file that presents the image for the advertisement. This element refers to the image that will be rendered in a browser.
- `NavigateUrl`: Specifies the URL of a page to navigate to, if a user clicks the advertisement image. If this parameter is not set, the ad is not "live." Although this parameter is optional, it must be specified, because the ad must direct clients to a target URL when it is clicked.
- `AlternateText`: Is an optional parameter that specifies some alternative text that will be displayed if the image specified in the `ImageUrl` parameter is not accessible. In some browsers, the `AlternateText` parameter appears as a ToolTip for the ad.
- `Keyword`: Is an optional parameter that specifies categories, such as computers, books, and magazines that can be used to filter for specific ads.
- `Impressions`: Is an optional parameter that provides a number that indicates the weight of the ad in the schedule of rotation relative to the other ads in the file. The larger the number, the more often the ad will be displayed.

KeywordFilter

The KeywordFilter property specifies a category filter to be passed to the source of the advertisement. A keyword filter allows the AdRotator control to display ads that match a given keyword. This enables the AdRotator control to display more context-sensitive ads, where the context is specified in the ASPX page containing the AdRotator control. When you use a keyword filter, three conditions arise:

- Both the `KeywordFilter` and `AdvertisementFile` properties are set. In such a case, the AdRotator control renders the image that matches the keyword specified.
- The `AdvertisementFile` property points to a valid Advertisement file, and the `KeywordFilter` property specifies a keyword that matches no images. In such a case, the control renders a blank image, and a trace warning is generated.

- The KeywordFilter property is empty. In such a case, keyword filtering will not be used to select an ad.

Target

The Target property specifies the name of the browser window or frame in which the advertisement needs to be displayed. This parameter can also take any of the HTML frame-related keywords, such as the following:

- _top: Loads the linked document into the topmost window.
- _blank: Loads the linked document into a new browser window.
- _self: Loads the linked document in the same window.
- _parent: Loads the linked document in the parent window of the window that contains the link.

After looking at the properties, let's understand the events associated with the AdRotator control.

Events of the AdRotator control

The AdRotator control supports the adCreated event that you can handle to monitor the activities of a user or a session. The adCreated event is generated with every round trip to the server, after the AdRotator control is created but before the page is rendered in the browser. The event handler for the adCreated event is OnAdCreated and has the following syntax:

OnAdCreated (sender as Object, e as AdCreatedEventArgs)

The event handler takes two parameters. The first parameter represents the object that raises the event. The second parameter represents the AdCreatedEventArgs object that contains the data related to this event. The AdCreatedEventArgs object has a set of properties that provide information specific to the AdCreated event:

- AdProperties: Is an IDictionary type object that provides all the advertisement properties that have been set for the currently selected advertisement.
- AlternateText: Is a String type value that sets the ALT property of the image that is sent to the browser. In some browsers, this text is displayed as a ToolTip when the mouse cursor hovers over the image.
- ImageUrl: Is a String value that sets the URL of the image that is displayed in the AdRotator control.
- NavigateUrl: Is a String type value that specifies the URL of the Web page to navigate to when a user clicks the advertisement.

The OnAdCreated event handler can be used to select ads in a local code or to modify the rendering of an ad selected from the Advertisement file. If an advertisement file is set, the parameters of the AdCreated event handler are set to the selected ad when the event is generated. The source image that is specified by the Advertisement file is sized by the browser to the dimensions of the AdRotator control, regardless of the image's actual size. The ad is selected based on impressions weighting from the file.
If the values are not set in the Advertisement file, the developer can modify the values in the ImageUrl, NavigateUrl, and AlternateText properties to modify the rendering of the AdRotator control. A very common use of this is when developers need to populate the event arguments with values pulled from a database.

Rendering ads to client browsers using AdRotator

The following code uses the AdRotator server-side control to render ads to the client browsers. The AdRotator control uses an Advertisement file named Ads.xml.

```
<%@ Page Language="VB" %>

<html>

   <head>

   </head>
```

```
    <body>
     <form runat="server">
      <h3><font face="Verdana">AdRotator Example</font></h3>
      <asp:AdRotator id="AdRotator1" runat="server" AdvertisementFile="Ads.xml"/>
     </form>

    </body>

</html>
```

The following code describes the Ads.xml file that is used by the AdRotator control. The file contains two advertisements that will be dynamically shown to different users. The first ad points to an image file named Saturn.gif. When users click this image, they are directed to the Saturn Web site. The second ad points to the image named Moon.jpg. When users click this image, they are directed to the Moon Web site.

```
<Advertisements>
  <Ad>
    <ImageUrl>
      saturn.gif
    </ImageUrl>

    <NavigateUrl>
      http://www.saturnrings.com/
    </NavigateUrl>

    <AlternateText>
     Saturn Rings Web Site
    </AlternateText>
    <Impressions>
      1
    </Impressions>
    <Keyword>
      Saturn
    </Keyword>
  </Ad>

  <Ad>
    <ImageUrl>
      Moon.jpg
    </ImageUrl>
    <NavigateUrl>
      http://www.moon.com
    </NavigateUrl>
    <AlternateText>
```

```
      Moon Explorers Web Site
    </AlternateText>
    <Impressions>
      1
    </Impressions>
    <Keyword>
      Moon
    </Keyword>
  </Ad>
</Advertisements>
```

Figure 4-1 shows the output of the preceding code.

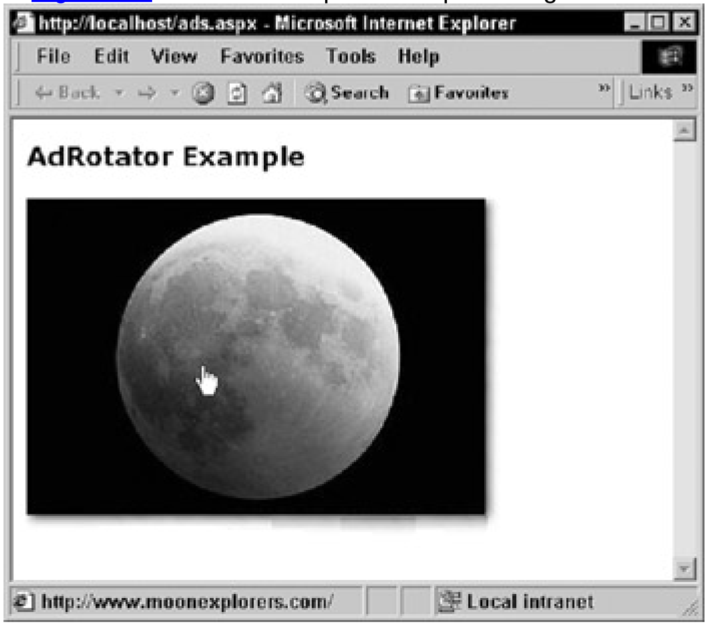

http://localhost/ads.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Links
AdRotator Example
http://www.moonexplorers.com/
Local intranet

Figure 4-1: Sample output of the AdRotator control

Using the Calendar Control

The Calendar control is used to display a one-month calendar. Users can use this control to view dates or select a specific day, week, or month.

The following is the syntax to add the Calendar control:

```
<asp:Calendar id="Calendar1" runat="server"
   propertyname = propertyvalue
   propertyname = propertyvalue
/>
```

Properties of the Calendar control

The Calendar control has properties that you can set when you add the control to your page. Table 4-1 describes some of the properties of the Calendar control.

Table 4-1: Properties of the Calendar control

Property	Description
`CellPadding`	Specifies the space between cells.
`CellSpacing`	Specifies the space

Table 4-1: Properties of the Calendar control

Property	Description
	between the contents of a cell and the cell's border.
`DayNameFormat`	Specifies the format of the day name.
`FirstDayOfWeek`	Sets a value for the day of the week that will be displayed in the calendar's first column.
`ShowNextPrevMonth`	Takes a Boolean value and specifies whether or not the calendar is capable of displaying next and previous month hyperlinks.
`NextMonthText`	Shows the HTML text for the "Next Month" navigation hyperlink if the `ShowNextPrevMonth` property is set to true.
`NextPrevFormat`	Specifies the format of the next month and previous month hyperlinks.
`PrevMonthText`	Shows the HTML text for the previous month hyperlink if the `ShowNextPrevMonth` property is set to true.
`SelectedDate`	Represents the date selected in the Calendar control.
`SelectedDates`	Specifies a collection of `DateTime` objects representing days highlighted on the calendar. This is a read-only property.
`SelectionMode`	Specifies whether the user can select a day, week, or month. The default is `Day`.
`SelectMonthText`	Shows the HTML text for the month selection in the selector column if the `SelectionMode`

Table 4-1: Properties of the Calendar control

Property	Description
	property is set to `DayWeekMonth`.
`SelectWeekText`	Shows the HTML text for the week selection in the selector column if the `SelectionMode` property is set to `DayWeek` or `DayWeekMonth`.
`ShowDayHeader`	Specifies whether or not to display the names of the days of the week.
`ShowGridLines`	Specifies a value that determines whether or not the days in the calendar should be displayed with gridlines around them. However, even if the property specifies to display lines around the calendar days, not all browsers can display the gridlines.
`TitleFormat`	Specifies the format of the month name in the title bar of the calendar.
`TodaysDate`	Specifies the current date.
`VisibleDate`	Specifies the month to be displayed in the calendar. The property is updated after the `VisibleMonthChanged` event is raised.

In addition to the properties in Table 4-1, the Calendar control has certain style objects associated with it. The style objects are used to set the appearance of the individual elements, such as the appearance of the day and week values of the control. Some style objects are described in Table 4-2.

Table 4-2: Style objects

Property	Description
`DayHeaderStyle`	Sets the appearance of the days of the current month.

Table 4-2: Style objects

Property	Description
`DayStyle`	Sets the appearance of the row above the calendar where the day names appear.
`NextPrevStyle`	Sets the appearance of the sections at the left and right ends of the title bar.
`OtherMonthDayStyle`	Sets the appearance of the days that are not in the displayed month.
`SelectedDayStyle`	Sets the appearance of the day selected by the user.
`SelectorStyle`	Sets the style properties for the week and month selector.
`TitleStyle`	Sets the appearance of the title bar at the top of the calendar containing the month name and month navigation links. If the value for `NextPrevStyle` is set, it overrides the extreme ends of the title bar.
`TodayDayStyle`	Sets the appearance of the current date.
`WeekendDayStyle`	Sets the

Table 4-2: Style objects

Property	Description
	appearance of the weekend days.

Events of the Calendar control

The Calendar control supports certain events that make the control interactive on the Web page. The supported events include the `DayRender`, `SelectionChanged`, and `VisibleMonthChanged` events. This section covers these events in detail.

DayRender event

The `DayRender` event is generated when a day cell is rendered. You can trap this event to modify the format and content of a particular day cell before the cell is rendered. The event handler for this event is `OnDayRender` and has the following syntax:

OnDayRender (sender as Object, e as DayRenderEventArgs)

The `DayRenderEventArgs` parameter contains data pertaining to this eve nt. This object has the following properties that can be used to make changes to the appearance of the day cell:

- `Cell`: Refers to a `TableCell` object that represents a table cell into which the day is rendered. A `TableCell` object has the following properties:
  - RowSpan: Represents the number of rows in the table that the cell spans.
  - `ColumnSpan`: Represents the number of columns in the table that the cell spans.
  - `HorizontalAlign`: Controls the horizontal alignment of the cell contents.
  - `VerticalAlign`: Controls the vertical alignment of the cell contents.
  - `Wrap`: Determines whether or not the contents wrap to fit the contents in the cell.
- `Day`: Refers to a `CalendarDay` object that represents the day being rendered. A `CalendarDay` object has the following properties:
  - `Date`: Represents the date, such as 15 July 2000, being rendered.
  - `DayNumberText`: Is a String that in turn represents the number of the day. For example, "15" is the `DayNumberText` for 15 July 2000.
  - `IsOtherMonth`: Is a Boolean value that returns True if the day cell being rendered is in the Calendar control's currently displayed month.
  - `IsSelectable`: Returns a Boolean value indicating whether or not the day cell being rendered can be selected.
  - `IsSelected`: Returns a Boolean value indicating whether or not the day cell being rendered is selected.
  - `IsToday`: Returns a Boolean value indicating whether or not the day cell being rendered is today's date.

- `IsWeekend`: Returns a Boolean value indicating whether or not the day cell being rendered is a Saturday or Sunday.

SelectionChanged event

The `SelectionChanged` event is generated when a user selects a day, week, or month by clicking the Calendar control. You can handle this event to validate against business logic the date selected by users. The event handler for this event is `OnSelectionChanged` and has the following syntax:

OnSelectionChange(sender As Object, e As EventArgs)
The `sender` parameter points to the control that generated this event, and any event-specific values are stored in the `EventArgs` object.

MonthChanged event

The `MonthChanged` event is generated when a user clicks the next or previous month navigation controls on the title heading of the Calendar control. The event handler for this event is `OnVisibleMonthChanged` and has the following syntax:

OnVisibleMonthChanged(sender as Object, e as MonthChangedEventArgs)
The `MonthRenderEventArgs` parameter contains data pertaining to this event. This object has the following properties that can be used to make changes to the appearance of the month:

- `NewDate`: Is a `DateTime` object that represents the new month that is selected.
- `PreviousDate`: Is a `DateTime` object that represents the previous month selected.

Rendering a Calendar to client browsers using the Calendar control

The following code uses the Calendar control to render a calendar in the client browsers:

```
<%@ Page Language="VB" %>
<html>
  <head>
   <script runat="server">
    Sub OnSelectionChanged (sender as Object, e as EventArgs)
      lblSelDate.Text = Calendar1.SelectedDate
    End Sub
   </script>
  </head>
  <body>
   <h3><font face="Verdana">Calendar control demo</font></h3>
   <form runat="server">
   <asp:Calendar id="Calendar1" runat="server"
    SelectionMode="DayWeekMonth"
    Font-Name="Verdana;Arial" Font-Size="12px"
    Height="180px" Width="230px"
    TodayDayStyle-Font-Bold="True"
    DayHeaderStyle-Font-Bold="True"
    OtherMonthDayStyle-ForeColor="gray"
    TitleStyle-BackColor="#3366ff"
    TitleStyle-ForeColor="white"
    TitleStyle-Font-Bold="True"
    SelectedDayStyle-BackColor="#ffcc66"
    SelectedDayStyle-Font-Bold="True"
    NextPrevFormat="ShortMonth"
    NextPrevStyle-ForeColor="white"
    NextPrevStyle-Font-Size="10px"
    SelectorStyle-BackColor="#99ccff"
    SelectorStyle-ForeColor="navy"
    SelectorStyle-Font-Size="9px"
    SelectWeekText = "wk"
    SelectMonthText = "month"
    OnSelectionChanged="OnSelectionChanged"
   />
   <BR>
   <asp:label style="font-name:Verdana;font-size:12px;forecolor:gray" id="lblSelDate"
runat="server"/>
   </form>
  </body>
</html>
```

Figure 4-2 shows the output of the preceding code. When you select a date, the date is displayed on the label.

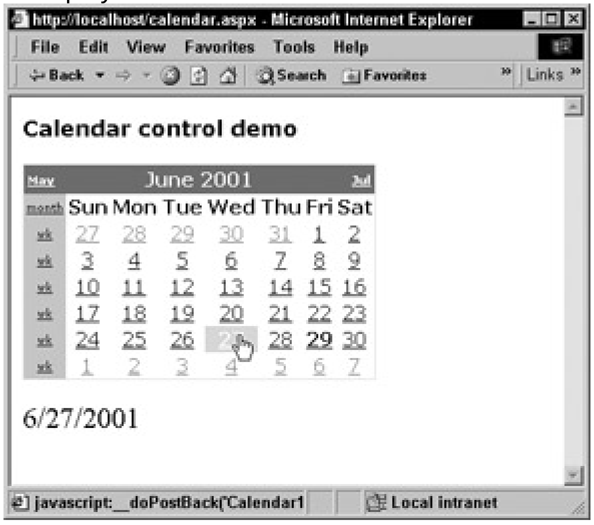

http://localhost/calendar.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Links
Calendar control demo
May June 2001 Jul
month Sun Mon Tue Wed Thu Fri Sat
wk 27 28 29 30 31 1 2
wk 3 4 5 6 7 8 9
wk 10 11 12 13 14 15 16
wk 17 18 19 20 21 22 23
wk 24 25 26 27 28 29 30
wk 1 2 3 4 5 6 7
6/27/2001
javascript:__doPostBack('Calendar1
Local intranet

Figure 4-2: Sample output of the Calendar control

Using the TreeView Control

The TreeView control is used to present hierarchical data to users in the Windows Explorer–style format, wherein the items can be expanded and collapsed. This control, like the other ASP.NET Server controls, is rendered as an HTML 3.2-compatible tree in older browser versions, such as Microsoft Internet Explorer 3.0. In newer browser versions, such as Microsoft Internet Explorer 5.5 and higher, this control is rendered by using the Dynamic HTML (DHTML) behaviors. Hence, compared to the older browser versions, the user experience is richer in the more recent browser versions.

Unlike the standard ASP.NET controls, TreeView and the other controls discussed in the sections to follow are not shipped as part of the ASP.NET Framework. These are additional controls that must be installed separately. Therefore, when you want to use these controls in an ASP.NET page, you must explicitly import the assemblies containing these controls. To import the assemblies, use the following code:

```
<%@import namespace="Microsoft.Web.UI.WebControls"%>

<%@Register TagPrefix="tp" Namespace = "Microsoft.

Web.UI.WebControls" Assembly="Microsoft.Web.UI.WebControls"%>
```

The `import` directive causes ASP.NET to import the contents of the specified namespace. The `Register` directive causes ASP.NET to identify all the controls in the specified assembly with the tag prefix "tp."

To add a TreeView control to the page, use the following syntax:

```
<tagprefix:TreeView runat="Server">

  <tagprefix:treenode text=".." DefaultStyle="  " HoverStyle="  "

SelectedStyle="  ">

<tagprefix:treenodetype Type="  " ChildType="  ">

    <tagprefix:treenode text="  "/>

<tagprefix:treenode text="  ">

</tagprefix:treenode>

</tagprefix:treenodetype>
```

The elements used in the preceding code are explained as follows:

- `TreeView`: Defines a TreeView control. It acts as a container for the nodes of the tree. The TreeView control is made up of various elements, every one of which is referred to as a *node*. Some nodes contain other nodes called *child* nodes. The container nodes are called *parent* nodes.
- TreeNode: Represents the node in the TreeView control.
- TreeNodeType: Defines the type of a node. A single TreeView control can have different types of nodes, such as a folder or any custom type.

Properties of the TreeView control

In addition to the properties that are inherited from the System.Web.UI.Control base class, the TreeView control has properties that can be used to control the behavior of the control. Some of these properties are described in Table 4-3.

Table 4-3: Properties of the TreeView control

Property	Description
`AutoPostBack`	Takes a Boolean value and indicates whether or not the control posts back to the server on each client request.
`AutoSelect`	Takes a Boolean value and indicates whether or not a tree node can be selected by simply pointing the mouse to the node, without having to click the

Table 4-3: Properties of the TreeView control

Property	Description
	node.
`DefaultStyle`	Sets a default style for the elements in the tree.
`ExpandedImageURL`	Sets an image to be displayed when a node is expanded.
`HoverStyle`	Sets a style, such as "font-family:Verdana;font-size:12pt;color:black," for the elements in the tree when the mouse hovers over them.
`ImageURL`	Sets an image to be displayed to represent a node.
`Indent`	Sets the number of pixels by which the child nodes need to be indented.
`ShowLines`	Takes a Boolean value and indicates whether or not lines are used to connect the nodes in the tree.

Events of the TreeView control
The events supported by the TreeView control include `Collapse`, `Expand`, and `SelectedIndexChanged`. The sections that follow look at each of these events in detail.

Collapse event

The `Collapse` event is generated when a user clicks a tree node to collapse it. You can trap this event to control the format and decide the contents of a particular node and its child nodes. The event handler for this event is `OnCollapse` and has the following syntax:

OnCollapse(sender As Object, e As TreeViewClickEventArgs)

As you can see, the event handler takes two arguments. The first argument, `As Object`, represents the object that generated the event. The second argument is the object of the TreeViewClickEventArgs class. This object contains the node information pertaining to this event. A `Node` object refers to the index of the node that was clicked, and has the following properties:

- `Expandable`: Sets or retrieves a value that indicates whether or not a plus-sign image is displayed with the node. A plus-sign image indicates that the node is expandable.
- Expanded: Indicates whether or not the node is expanded.
- Level: Returns the level of the node; level 0 refers to the root.
- Text: Returns the text of the selected node.

Expand event

The `Expand` event is generated when a user clicks a tree node to expand it. You can trap this event to control the formatting and decide the contents of a particular node and its child nodes. The event handler for this event is `OnExpand` and has the following syntax:

OnExpand(sender As Object, e As TreeViewClickEventArgs)
The second parameter is an object of the `TreeViewClickEventArgs` class and contains the data pertaining to the `Expand` event.

SelectedIndexChanged event

The `SelectedIndex` event is generated when a user clicks the TreeView control to change the active tree node. This causes the TreeView control to move the highlight from the node that was selected earlier to the newly selected node. You can trap this event to control the formatting and decide the contents of the selected node. The event handler for this event is `OnSelectedIndexChanged` and has the following syntax:

OnSelectedIndexChanged(sender As Object, e As TreeViewSelectEventArgs)
The second parameter is the `TreeViewSelectEventArgs` object and contains the data pertaining to the `SelectedIndexChanged` event. This object has the following properties that can be used to make changes to the appearance of the selected node:

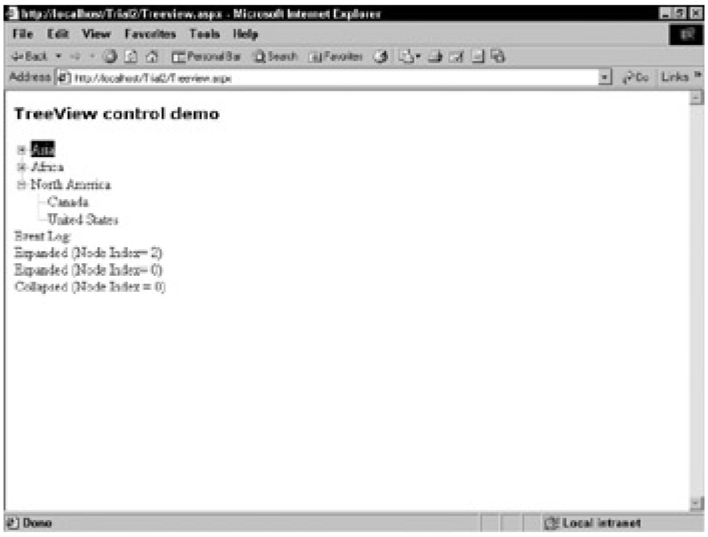
- `NewNode`: Refers to a `Node` object that represents the tree node that has been selected.
- OldNode: Refers to a Node object that represents the tree node that was previously selected.

Rendering a TreeView control

The following code renders a TreeView control in a page:

```
<%@ Page Language="VB" %>
<%@import namespace="Microsoft.Web.UI.WebControls"%>
<%@Register TagPrefix="mytree" Namespace = "Microsoft.
Web.UI.WebControls" Assembly="Microsoft.Web.UI.WebControls"%>
<html>
  <script language="VB" runat="server">
   Sub OnCollapse( sender as Object, e as TreeViewClickEventArgs)
     'append node index to the label control when tree is
     'collapsed
     mylabel.Text += "<BR>Collapsed (Node Index = " & e.Node.ToString() + ")"
   End Sub

   Sub OnExpand (sender as Object , e as TreeViewClickEventArgs )
     ' append node index to label control when tree is
     'expanded
     mylabel.Text += "<BR>Expanded (Node Index= " & e.Node.ToString() + ")"
   End Sub
```

```
    Sub OnSelectedIndexChanged ( sender as Object,  e as TreeViewSelectEventArgs)
      ' append node index to label control when a new node is
      'selected in the tree
      mylabel.Text += "<BR>Selected " & e.NewNode.ToString() & " (oldNode Index=" +
e.OldNode.ToString()+")"
    End Sub
   </script>
<head>
</head>

<body>

   <h3><font face="Verdana">TreeView control demo</font></h3>
   <form runat="server">

   <! — render tree view control, setup event handlers for collapse, expand and
selectedindexchanged events -->

   <mytree:TreeView runat="server" AutoPostBack="true"
DefaultStyle="font-name:Verdana;font-size:12pt;color:black;" SelectedStyle="
font-face:Verdana;font-size:12pt;color:white;" OnCollapse="OnCollapse"
OnExpand="OnExpand"
OnSelectedIndexChanged="OnSelectedIndexChanged">
     <mytree:treenode text="Asia">
       <mytree:treenode text="China"/>
       <mytree:treenode text="India"/>
     </mytree:treenode>
     <mytree:treenode text="Africa">
       <mytree:treenode text="Zaire"/>
       <mytree:treenode text="Zambia"/>
     </mytree:treenode>
     <mytree:treenode text="North America">
       <mytree:treenode text="Canada"/>
       <mytree:treenode text="United States"/>
     </mytree:treenode>
   </mytree:treeview>
<br>
   <asp:label id=mylabel runat="server">Event Log: </asp:label>
   </form>
</body>
</html>
```
Figure 4-3 shows the output of the preceding code.


http://localhost/Trial2/Treeview.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Personal Bar Search Favorites
Address http://localhost/Trial2/Treeview.aspx
Go Links
Done
Local intranet

```
TreeView control demo

⊞ Asia
⊞ Africa
⊟ North America
    Canada
    United States
Event Log:
Expanded (Node Index= 2)
Expanded (Node Index= 0)
Collapsed (Node Index = 0)
```

Figure 4-3: Sample output of a TreeView control

Using the TabStrip and MultiPage Controls

The TabStrip control is used to present tabbed controls, which can be used along with the MultiPage control to display varied information in a given space. The TabStrip control renders tabs that users can click to switch between the different tabs. The MultiPage control is used to display multiple pages of data in a given screen area. This control is typically used with the TabStrip control.

TabStrip control

You use the following syntax to add a TabStrip control to a page:

```
<tagprefix:TabStrip runat="Server" TabDefaultStyle=".." TabHoverStyle=
".." TabSelectedStyle=".." SepDefaultStyle="..">


  <tagprefix:Tab text=".." >
    <tagprefix:Tab text="Node1.1"/>
    <tagprefix:Tab text="Node1.2">
  </tagprefix:Tab>
</tagprefix:TabStrip>
```

The TabStrip control uses the following elements to define the tabbed interface to be rendered:

- `TabStrip`: Defines a TabStrip control, which acts as a container for the tabs and tab separators.
- `Tab`: Defines a tab element in the TabStrip control, which is rendered on the client browser as tabs on top of the tab strip.
- `TabSeparator`: Represents the separator bars between the tabs.

Table 4-4 describes some of the properties of the TabStrip control.

Table 4-4: Properties of the TabStrip control

Property	Description
AutoPostBack	Specifies whether or not the control

Table 4-4: Properties of the TabStrip control

Property	Description
	posts back to the server on every client request.
DefaultStyle	Specifies the default style of the TabStrip control.
Orientation	Specifies the orientation of the tabs, which can be horizontal or vertical.
SelectedIndex	Returns the index of the selected tab.
SepDefaultStyle	Specifies the default style for the tab separators.
SepHoverStyle	Specifies the style to be applied to the tab separators when the mouse hovers over the separators.
TargetID	Specifies the name of the MultiPage control to which the tabs will be linked automatically.

The TabStrip control supports the SelectedIndexChanged event, which is fired when a user shifts from one tab to another. This event can be trapped to control the formatting and decide the contents of a particular tab. The event handler for this event is OnSelectedIndexChanged and has the following syntax:

```
OnSelectedIndexChanged(sender As Object, e as EventArgs)
```

The second parameter is the EventArgs object and contains data pertaining to this event.

MultiPage control

The MultiPage control is a container control that contains a set of PageView elements, which are used to render different pages in a given screen space. The PageView elements contain the visible part of the MultiPage control. The MultiPage control is typically used with the TabStrip control to give users the ability to navigate from one page to another.

The following code segment creates a MultiPage control with two PageView elements:

```
<tagprefix:MultiPage runat="server" selectedindex="1">
  <tagprefix:PageView>
    <P> Data for page view <B>1</B> </P>
  </tagprefix:PageView>

  <tagprefix:PageView>
    <P> Data for page view <B>2</B> </P>
  </tagprefix:PageView>

</tagprefix:MultiPage>
```

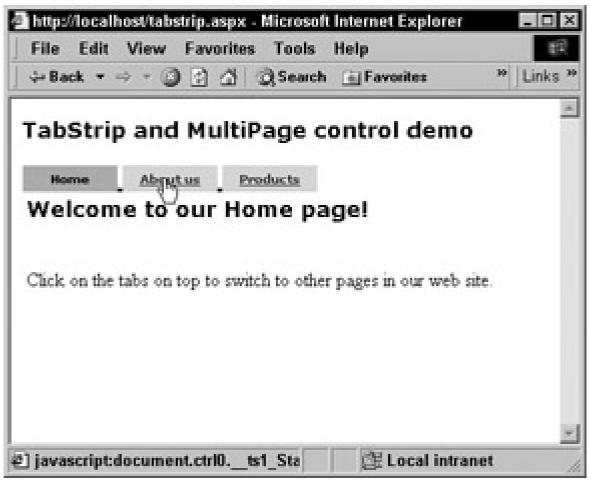
Just like the TabStrip control, the MultiPage control supports the SelectedIndex property, which indicates the selected PageView.

Using MultiPage and TabStrip controls together

As mentioned, the TabStrip control provides navigation capabilities and the MultiPage control provides the ability to view multiple pages in the same screen area. The two controls typically are used in combination.
To actually combine the MultiPage control with the TabStrip control, you need to set the TargetID property of the TabStrip control to the ID of the MultiPage control. This enables the TabStrip control to automatically switch from one PageView element to another when a user clicks a tab.

The following code renders the TabStrip and MultiPage controls on the page:

```
<%@ Page Language="VB" %>
<%@import namespace="Microsoft.Web.UI.WebControls"%>
<%@Register TagPrefix="myts" Namespace = "Microsoft.Web.
UI.WebControls" Assembly="Microsoft.Web.UI.WebControls"%>
<html>
<head>
</head>
<body>
  <h3><font face="Verdana">TabStrip and MultiPage control demo</font></h3>

  <! — render the TabStrip control and set the TargetID to point
to the multipage control-->

  <form runat="server">
  <myts:TabStrip id="ts1" runat="server"
TabDefaultStyle="background-color:lightgrey;font-family:verdana;
font-weight:bold;font-size:8pt;color:blue;width:79;height:21;text-align:center"
TabHoverStyle="background-color:#777777"
TabSelectedStyle="background-color:darkgray;color:#000000"
SepDefaultStyle="background-color:#FFFFFF;border-color:darkblue;border-width:
3px;border-style:solid;border-top:none;border-left:none;border-right:none" TargetID="
mymultipage">

    <myts:Tab Text="Home" />
    <myts:TabSeparator/>
    <myts:Tab Text="About us" />
    <myts:TabSeparator/>
    <myts:Tab Text="Products" />
  </myts:TabStrip>

  <! — render the MultiPage control and notice that the id of the control has been
set as the targetID of the TabStrip control-->

  <myts:MultiPage id="mymultipage" runat="server">
    <myts:pageview><P><H3 style="font-family:verdana"> Welcome to our Home page!
</H3>
<br> Click on the tabs on top to switch to other pages in our web
site.</P></myts:pageview>

    <myts:pageview><P><H3 style="font-family:verdana"> About Us  </H3></P>
</myts:pageview>

    <myts:pageview><P><H3 style="font-family:verdana"> Product Information here
</H3>
</P>
</myts:pageview></myts:multipage>
  </form>
</body>
</html>
```
The output of the preceding code is shown in Figure 4-4.


http://localhost/tabstrip.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Links
Home
About us
Products
javascript:document.ctrl0.__ts1_Sta
Local intranet

TabStrip and MultiPage control demo

Welcome to our Home page!

Click on the tabs on top to switch to other pages in our web site.

Figure 4-4: Sample output of the TabStrip and MultiPage controls

Using the Toolbar Control

The Toolbar control is used to render a toolbar in the client browsers. At the simplest level, a toolbar is a collection of graphical buttons. The Toolbar control is typically used to provide the commonly used functionality to users in a graphical form.

To add the Toolbar control to a page, use the following syntax:

```
<tagprefix:Toolbar ..>
  <tagprefix:ToolbarButton Text=".." ImageUrl=".." />
  <tagprefix:ToolbarSeparator />
  <tagprefix:ToolbarButton Text=".." ImageUrl=".."/>
  <tagprefix:ToolbarButton Text=".." ImageUrl=".."/>
</tagprefix:Toolbar>
```

As you can see in the preceding syntax, the Toolbar control is a container control that contains elements to define a toolbar. These elements are described as follows:

- `ToolbarButton`: Defines a button on the toolbar.
- `ToolbarCheckButton`: Defines a check button on the toolbar.
- `ToolbarCheckGroup`: Defines a group of check buttons on the toolbar.
- `ToolbarLabel`: Defines a label to display plain text on the toolbar.
- `ToolbarSeparator`: Defines a separator on the toolbar, which is useful in identifying the separate groups of toolbar buttons.
- `ToolbarTextBox`: Defines a text box on the toolbar.

Properties of the Toolbar control

In addition to the properties that are inherited from the System.Web.UI.Control base class, the Toolbar control has additional properties. Table 4-5 describes some of these properties.

Table 4-5: Properties of the Toolbar control

Property	Description
AutoPostBack	Specifies whether or not the control posts back to the server on every

Table 4-5: Properties of the Toolbar control

Property	Description
	client request.
DefaultStyle	Specifies the default style of the toolbar.
HoverStyle	Specifies the style to be applied when the mouse hovers over the toolbar.
SelectedStyle	Specifies the style to be applied when the toolbar items are selected
Orientation	Specifies the orientation of the toolbar, which can be horizontal or vertical.

Note Every button on the toolbar has three states — Default, Selected, and Hover. You can define appropriate CSS styles for each of the three states. Then, ASP.NET will apply the appropriate style when rendering the button.

Events of the Toolbar control

The Toolbar control supports the ButtonClick and CheckChange events, which make the control interactive when rendered on a page. The following sections look at each of the events in detail.

ButtonClick event

The ButtonClick event is generated when a user clicks a toolbar button. The event handler for this event is OnButtonClick and has the following syntax:

```
OnButtonClick(sender As Object, e As EventArgs)
```

The second parameter is the EventArgs object and contains the data pertaining to this event. To retrieve the toolbar data in the event handler, the sender variable must be converted into a variable of type ToolbarButton. To see how the data is converted, see the example of the toolbar control in the upcoming section "Rendering a toolbar."

CheckChange event

The CheckChange event is generated when the state of a ToolbarCheckButton changes. This event is trapped to respond to any change in the state of a ToolbarCheckButton. Here is the event handler for this event:

OnCheckChange(sender As Object, e As EventArgs)
The EventArgs parameter contains data pertaining to this event. To retrieve the toolbar data in the event handler, the sender variable must be converted into a variable of type ToolbarButton. The syntax for the same is given as follows:

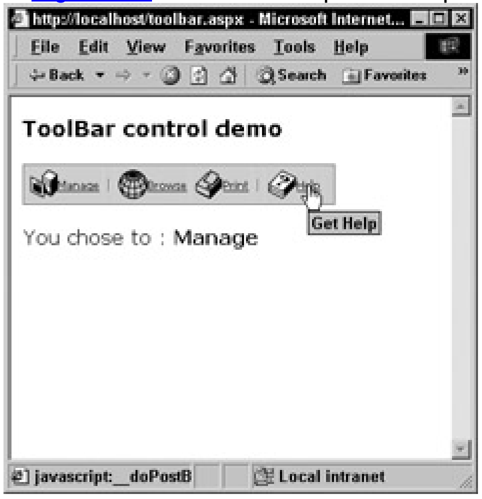
```
Dim tb as ToolbarButton
tb=CType(sender,ToolbarButton)
```

Rendering a toolbar

The following code example renders a toolbar on the page:

```
<%@ Page Language="VB" %>
<%@import namespace="Microsoft.Web.UI.WebControls"%>
<%@Register TagPrefix="ie" Namespace = "Microsoft.Web.UI.
WebControls" Assembly="Microsoft.Web.UI.WebControls"%>
<html>

  <script runat="server" language="VB">
    sub OnButtonClick(sender as object, e as EventArgs)
       Dim sMsg as String, tb as ToolbarButton
       'convert from Object type to ToolbarButton type
       tb=CType(sender,ToolbarButton)
       sMsg="<BR>You chose to : <B>" & tb.Text & "</B>"
       lblMessage.Text = sMsg
    End Sub
  </script>

<head>
</head>
<body>
  <h3><font face="Verdana">ToolBar control demo</font></h3>
  <! — display toolbar control, setup event handler for Buttonclick Event-->
  <form runat="server">
  <ie:Toolbar id="tb2" runat="server" BorderColor="Gray"
Font-Name="Tahoma" Font-Size="8pt" BackColor="#CCCCCC" Width="75%"
OnButtonClick="
OnButtonClick">
      <ie:ToolbarButton Text="Manage" ImageUrl="mmc.gif" Tooltip="Manage Server"/>
      <ie:ToolbarSeparator />
      <ie:ToolbarButton Text="Browse" ImageUrl="web.gif" Tooltip="Browse Info"
selectedstyle="color:red;font-size:12pt;"/>
      <ie:ToolbarButton Text="Print" ImageUrl="print.gif" Tooltip="Print Document" />
      <ie:ToolbarSeparator />
      <ie:ToolbarButton Text="Help" ImageUrl="help.gif" Tooltip="Get Help" />
  </ie:Toolbar>
  <asp:label id=lblMessage runat="server" style="font-family:verdana"/>
  </form>
</body>
</html>
```

Figure 4-5 shows the output of the preceding code.


http://localhost/toolbar.aspx - Microsoft Internet...
File Edit View Favorites Tools Help
Back Search Favorites
Manage Browse Print Help
Get Help
javascript:__doPostB
Local intranet

ToolBar control demo

You chose to : Manage

Figure 4-5: Sample output of the Toolbar control

Summary

In this chapter, you learned about the functionality of the Rich Web controls. First, you learned the properties, methods, and events associated with the AdRotator and Calendar controls. Then, you learned how to create and use the additional Rich Web controls, such as TreeView, TabStrip, MultiPage, and Toolbar. You learned how to set their properties and handle the events raised by them.

Chapter 5: Creating and Using Custom Controls

Overview

Visual Studio .NET provides a rich set of standard controls, which are also called intrinsic controls. These standard controls provide a wide range of functionality and allow you to design and develop a user-friendly interface for your applications. Additionally, Visual Studio .NET provides you custom controls that you can use if the existing controls do not meet your requirements. For example, consider a Web application that needs to have multiple Web Forms and most of the Web Forms need a calculator. In this case, instead of adding standard controls to each Web Form for implementing the calculator, you can create one custom control to represent a calculator and use it across the Web Forms in your application. Thus, custom controls allow reusability. This chapter describes the procedure to create and use custom controls.

Introduction to Custom Controls

You can create the following types of custom controls in Visual Studio .NET:

- **User control:** A Web Forms page that can be used as a control on other Web Forms pages. Thus, if you already have a Web Forms page and you need to construct a similar one, you can use the existing page as a user control.
- **Composite control:** A combination of existing controls that is used as a single control. You can create a composite control in any .NET

programming language and use it on an ASP.NET page. For example, you can create a composite control comprising a button and a text box in C# and use it on an ASP.NET page.

In addition to the custom controls discussed, you can perform the following actions with controls:

- Extend the functionality of the existing Web Form controls to meet your requirements. For example, consider a situation in which an existing Web Forms control meets almost all of your requirements, but you need some additional features in the control. In such a situation, you can add more features to your Web Form and customize the control. This can be done by inheriting from the control and overriding its properties, methods, or events.
- Develop a custom control by inheriting directly from one of the Control base classes. You'll need to do this when none of the existing Web Forms controls meets any of your requirements. The benefit of using the existing classes to create custom controls is that they provide the basic framework needed by a Web Forms control. This way, you can concentrate more on programming the features that you want to incorporate.

Before you create your own custom controls, let us examine the base classes used by the controls.

Basic Structure of Web Forms Controls

This section equips you with the basic understanding of the elements involved in developing a Web Forms control. We will first discuss the classes that are used to create Web Forms. This is followed by a discussion of the interfaces that can be implemented in Web Forms controls.

Classes used for Web Forms controls

Each Web Forms control is a class that inherits from the System.Web.UI. Control class directly or indirectly. Therefore, in this section, we examine the System.Web.UI.Control class and its inherited classes that are used to create a Web Forms control.

The System.Web.UI.Control class

The System.Web.UI.Control class defines all the properties, events, and methods that are common to all the Web Forms controls. You need to inherit your control from this class in the following cases:

- When your control does not have a user interface
- When your control includes other controls that render their own user interface

Some of the properties, methods, and events of the Control class are described in Table 5-1, Table 5-2, and Table 5-3 respectively.

Table 5-1: Control properties

Property	Description
`ID`	Represents the control identifier to refer to the server control in programs.
`Parent`	Represents the parent

Table 5-1: Control properties

Property	Description
	control in the server control hierarchy.
`Visible`	Indicates whether or not a server control should be rendered on the page.

Table 5-2: Control methods

Method	Description
`Dispose`	Causes a server control to perform final cleanup.
`Equals`	Used to check whether or not an object is the same as the current object. This method is overloaded.
`FindControl`	Used to search a container for a specified server control. This method is overloaded.
`ToString`	Used to return the string representati on of the current object.

Table 5-3: Control events

Event	Description
`Init`	Is fired when a control is initialized. This is the

Table 5-3: Control events

Event	Description
	first step when a page needs to be displayed in a browser.
`Load`	Is fired when the control is loaded in a page.
`Unload`	Is fired when a control is unloaded from the memory.

The System.Web.UI.WebControls.WebControl class

The System.Web.UI.WebControls.WebControl class is the base class for all Web controls. This class provides properties and methods to implement user interface functionality. It is inherited from the Control class. Some of the properties that are used to render additional user interface functionality are `ForeColor`, `BackColor`, `BorderStyle`, `Width`, and `Height`. Web controls, such as Label, TextBox, Button, Table, and Calendar, all inherit from the WebControl class. Therefore, if you have a control that has a user interface, it should inherit from the WebControl class.

The System.Web.UI.HtmlControls.HtmlControl class

The HtmlControl class is the base class for all HTML controls in Web Forms. This class inherits from the Control class. The controls provided by this class map directly to HTML elements. Therefore, these controls are useful for migrating ASP applications to ASP.NET.

Interfaces used for Web Forms controls

Several interfaces are available that you can implement depending upon your requirements. For example, if your control provides data binding, you need to implement the INamingContainer interface. In this section, we examine the interfaces that you might need to implement when you create controls.

Note Interfaces are the collection of properties, methods, and events that are implemented through classes.

INamingContainer interface

This interface is used when you need to create controls that satisfy any of the following conditions:

- Provides data binding
- Is a templated control
- Routes events to its child controls

The INamingContainer interface doesn't have methods. When this interface is implemented by a control, the ASP.NET page framework creates a namespace for the control and ensures that each child control in the parent control has a unique ID.

IPostBackDataHandler interface

A control should implement the System.Web.UI.IPostBackDataHandler interface when it needs to update its state or raise events on the server after examining the postback data.
For example, the data sent to a TextBox control might result in the text box changing its text, as determined by its `Text` property. When the text changes, the text box also raises a `TextChanged` event. Thus, this control is examining the data, changing its state, and raising events.

IPostBackEventHandler interface

The IPostBackEventHandler interface is implemented in a control when you want to transfer events that are generated on the client side to the server. Events generated on the client side are postback events, hence the interface name.
An example of implementing this interface is when a user submits a form on the client side. In this case, the server does not know that the form is submitted unless the IPostBackEventHandler interface generates the `Click` event on the server when the form is submitted. This ensures that the server is in sync with the events that occur at the client end.

Creating Custom Controls

In this section, you'll create a custom control that represents a product form. The form provides text boxes to enter the product ID and product name of a specific product. You can reuse this control in the pages that need user input for any product. Figure 5-1 displays a sample user control.

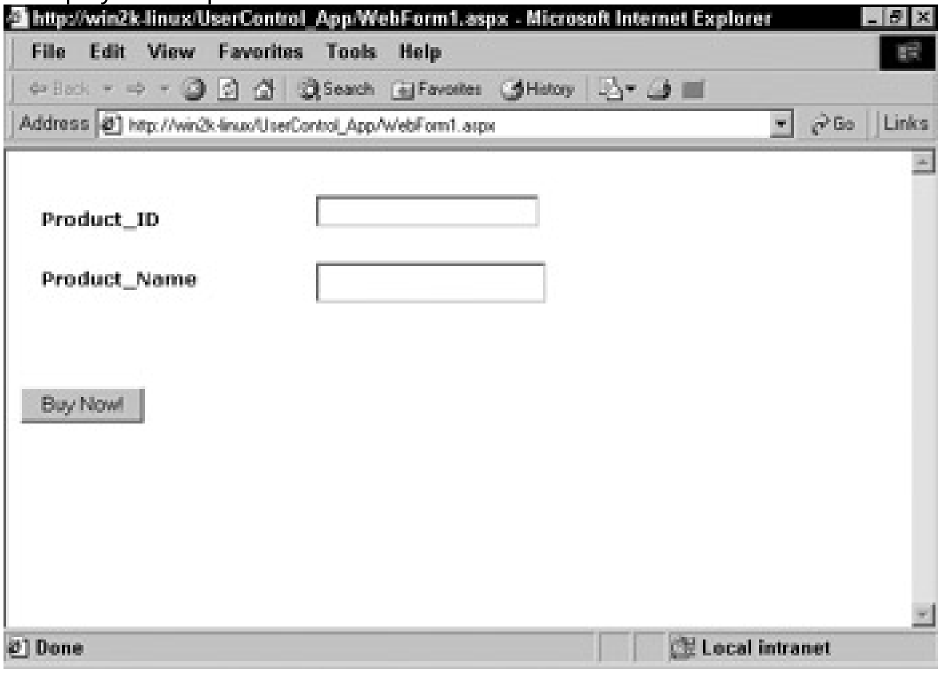

http://win2k-linux/UserControl_App/WebForm1.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://win2k-linux/UserControl_App/WebForm1.aspx
Go Links
Product_ID
Product_Name
Buy Now!
Done
Local intranet

Figure 5-1: A sample user control

Let us now create this sample control.

Creating and using a user control
You can create a user control by creating a Web Forms page and excluding its `<HTML>`, `<HEAD>`, `<BODY>`, and `<FORM>` elements in the page. Let us now create a Web Forms page and convert it into a user control. The Web Forms page that you convert to a user control can be designed in either Visual Basic or C#.

Creating a user control

Creating a user control involves designing a Web Forms page to work as a control and then modifying the extension of the Web Forms file to .ascx. The steps are described in the sections that follow.

Step 1: Add a Web Form to the existing project

The first step involves adding a Web Form to the existing project. To do so:

1. Select the Add Web Form option from the Project menu. The Add New Item dialog box appears.
2. Type the name of the Web Form (for example, MyWebForm) in the Name box.

   Note The Add New Item dialog box already displays a default name for the Web Form. You can modify it, if necessary.
3. Click Open. This will open a new Web Form.

Step 2: Convert your Web Form to a user control

This step involves converting your Web Form to a user control. You can do so by adding the controls to the form using the Toolbox and then editing the HTML code of the ASPX file. In our example of the Web Form displayed in Figure 5-1, the Toolbox has been used to add controls to the Web Form. As you can see, the Web Form has two text boxes, labeled Product_ID and Product_Name. The HTML code is automatically generated when you add these text boxes and labels.

After creating the visual interface for the user control, you need to edit the HTML file. As discussed earlier, the user control HTML file cannot contain the HTML tags that include the `<HEAD>`, `<BODY>`, `<HTML>`, and `<FORM>` tags. Therefore, you need to remove these tags from the HTML file. Additionally, you also need to change the `@ Page` directives to `@ Control` directives.

After you remove the HTML tags, your HTML file should look like this:

```
<table height=90 cellspacing=0 cellpadding=0 width=361 border=0
ms_2d_layout="TRUE">
  <tr valign=top>
    <td width=14 height=15></td>
    <td width=188></td>
    <td width=159></td></tr>
  <tr valign=top>
    <td colspan=2 height=9></td>
    <td rowspan=2>
<asp:TextBox id=Product_ID runat="server"></asp:TextBox></td></tr>
  <tr valign=top>
    <td height=37></td>
    <td>
<asp:Label id=Label1 runat="server" font-bold="True" font-names="Verdana" font-size=
"Smaller">Product_ID</asp:Label></td></tr>
  <tr valign=top>
    <td colspan=2 height=4></td>
    <td rowspan=2>
<asp:TextBox id=Product_Name runat="server" Width="158"
Height="28"></asp:TextBox></td>
```

</tr>

  <tr valign=top>

    <td height=25></td>

    <td>

<asp:Label id=Label2 runat="server" Width="133" Height="19" font-bold="True" font-names=

"Verdana" font-size="Smaller">Product_Name</asp:Label></td></tr></table>

Note The table elements will appear only if the layout of the page is changed to GridLayout.

After editing the code, save and close the file.

Step 3: Change the extension of the file to .ascx

This step involves changing the extension of the file to .ascx. To rename the file, you need to follow these steps:

1. Right-click the user control Web Form file in the Solution Explorer window and select Rename from the shortcut menu.
2. Change the extension of the file to .ascx.

Note Do not leave the user control file open while renaming it. If the user control file is open, you will not be able to rename the file.

Using a user control in a Web Forms page

After you create a user control, you can use it on another Web Forms page. To do so, you need to register the control and then add it to your Web Forms page. The steps are described as follows.

Step 1: Register the user control

This step involves registering the user control that you created. To do so, you'll need to follow these steps:

1. Open the Web Forms page in which you want to add your user control. Open the WebForm1.aspx file that was created by default when you created the Web application project.
2. Write the following code:
3. <%@ Register TagPrefix="Acme" TagName="Product" Src="MyWebForm.ascx" %>

This code will register the MyWebForm.ascx file. `TagPrefix` is an alias name that is used to identify the namespace on the Web Forms page to which it is added. The `TagName` tag contains the alias name for the class that represents the user control. `Src` is the name of the file that has the user control to be registered. The user control file can be within the same project or another project.

Step 2: Add the user control to your Web Forms page

This step involves adding the user control to your Web Forms page. To do so, use the following code:

<Acme:Product id="MyProduct" runat="Server"/>

The code given needs to be placed in the script where you want the control to appear on the page. In most cases, the code is placed in the `<BODY>` region of the page within the `<FORM>` element. In this code:

- Acme is the `TagPrefix` for your user control.
- MyProduct is the ID of your user control. You'll use this ID to refer to the control in programs.

Your Web Forms page will display the user control when you run the program.

Developing a composite control

You can create new controls using one or more existing controls. Such controls that aggregate a number of controls are referred to as composite controls. The primary difference between composite controls and user controls is that composite controls are compiled into assemblies and the compiled file is included into the project in which you want to include the control. In this section, we will create a composite control in C# and use it on an ASP page.

Concepts involved in creating a composite control

When you create a composite control, you need to do the following:

- Define the class that implements the composite control. The class needs to inherit from the Control class and, optionally, the INamingContainer class.
- Optionally override the CreateChildControls method of the Control class. The CreateChildControls method creates any child controls that need to be drawn on a page. This method is used in composition-based rendering, wherein child controls are instantiated and rendered on the page.
- Optionally implement the Render method of the Control class. You need to implement this method when you use the rendering logic instead of composition to render the ASP page. When you render controls, the performance overhead is less because controls need not be instantiated. Instead, the page is rendered as defined by the Render method. The Render method controls the output of the page at run-time.

Creating the control

You have examined the basic concepts to create a control. Let us now create a composite control. We will create the control in C#. The same programming logic can be used in VB.NET as well, except that the syntax will change.

The control that we will create comprises a Calendar control, a TextBox control, a Submit button, and a Label control. The user is expected to select his or her date of birth from the calendar, specify their work experience in years, and click the Submit button to ascertain whether he or she is eligible for a job.

To create the custom control project, create a new Class Library project in C#.

The controls that you need to use on the form are in the System.Web namespace. Therefore, include a reference to the System.Web.dll file. To include the reference, you need to perform the following steps:

1. Select the Add Reference option from the Project menu. The Add Reference dialog box appears.
2. In the Add Reference dialog box, from the .NET tab, select System.Web.dll and click Select.
3. The component moves to the Selected Components list. Click OK to add the reference.

After you add a reference to the System.Web.dll file, you can write the code for the control. To code the control, select the class module from the Solution Explorer. In the class module, declare the namespaces that are used by the control by specifying the following statements:

```
using System;

using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
```

Next, declare the namespace for the control and declare a public class in the namespace. You should also declare the properties that you need to expose for the control. In the following code, we have declared the namespace, class, and a `Text` property for an EmpElg label. Additionally, we have also left placeholders for the `CreateChildControls` method and the `Click` event of the Submit button.

```
namespace CalcControl
{
  /// <summary>
  /// This class is used to establish if an applicant is
 ///eligible for job
  /// </summary>
  public class CalcClass : Control, INamingContainer
  {
    private Label EmpElg;
    public string Text
    {
      get
      {
        EnsureChildControls();
        return EmpElg.Text;
      }
      set
      {
        EnsureChildControls();
        EmpElg.Text=value;
      }
    }
    protected override void CreateChildControls()
    {
    }
    protected void Submit_Click(object sender, System.EventArgs e)
    {
    }
  }
}
```

The following is the code for the `CreateChildControls` method. In this code, we are declaring a few controls for the form and we are also using literal controls to display plain text on the form.

```
protected override void CreateChildControls()
{
  Controls.Add(new LiteralControl("<h3>Select date of birth : "));
  Calendar Cal1 = new Calendar();
  //Cal1.TodaysDate();
  Controls.Add(Cal1);
  Controls.Add(new LiteralControl("<h3>Work Experience (Years) :
       "));
  TextBox WorkEx = new TextBox();
  WorkEx.Text="0";
  Controls.Add(WorkEx);
  Controls.Add(new LiteralControl("</h3>"));
  Button Submit = new Button();
  Submit.Text = "Submit";
  Controls.Add(new LiteralControl("<br>"));
  Controls.Add(Submit);
  Submit.Click += new System.EventHandler(this.Submit_Click);
  Controls.Add(new LiteralControl("<br><br>"));
  EmpElg = new Label();
  EmpElg.Height = 50;
  EmpElg.Width = 500;
  EmpElg.Text = "Check your eligibility.";
  Controls.Add(EmpElg);
}
```

Finally, the code for the Submit button, which is used to check the eligibility of an employee, is as follows:

```
protected void Submit_Click(object sender, System.EventArgs e)
{
  EnsureChildControls();
  if (Int32.Parse(((TextBox)Controls[3]).Text)>=5)
  {
    if ((((Calendar)Controls[1]).SelectedDate.Year) <= 1975)
    {
      EmpElg.Text = "You are eligible to apply for a job in our
       company!!";
    }
    else
    {
      EmpElg.Text = "You are NOT eligible to apply for a job in
       our company!!";
    }
  }
  else
  {
    EmpElg.Text = "You are NOT eligible for applying for a job in
       our company!!";
  }
}
```

When the user clicks Submit, this code checks whether the work experience of the user is more than five years. It also checks whether the user is born in or before 1975. When both the conditions are satisfied, the user is considered eligible for the job.

Note You can find the complete code for creating a composite control on the companion Web site for this book.

Compile the application to create the DLL file for the composite control. After compiling the file, you can proceed and include the file on an ASP page and check whether the control works as desired.

Adding the composite control to a page

After you have compiled the composite control into a DLL, you can include it in a Web application. The steps to include the control into a Web application are given as follows:

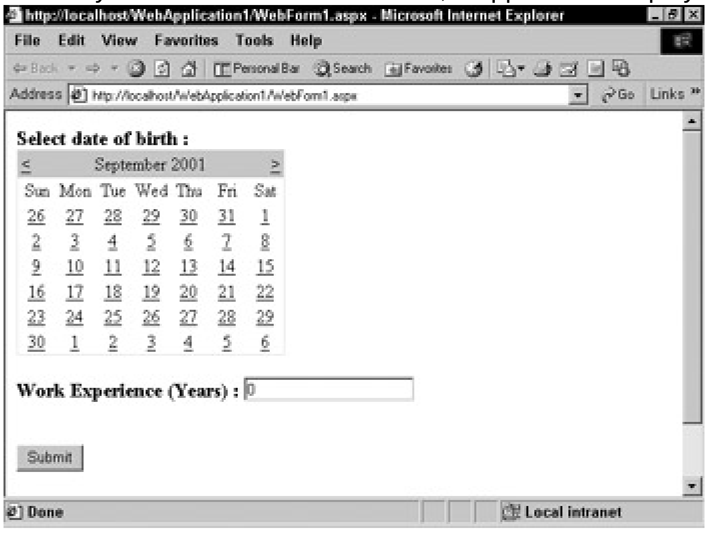
1. Create a new ASP Web application or open an existing project.
2. Add a reference to the custom control that you created in the previous step. To add a reference, in the Add Reference dialog box, select the Projects tab and browse to the DLL file of your custom control.
3. In the HTML source file, specify the `@ Register` directive to register the control. For example, if the name of the control namespace is CalcControl and the name of the class library project is CustomControls, you can register the control by specifying the following statement:

```
4.    <%@ Register TagPrefix="Custom" Namespace="CalcControl"
        Assembly =
"CustomControls" %>
```

5. Include the control on the page by using the tag name with which the control was registered. For example, in the preceding case, the tag name is Custom. Therefore, to include the control in the `<BODY>` region of the form, write the following code:

```
6.    <form id="Form1" method="post" runat="server">
7.      <Custom:CalcClass id="CalcClass" Text="Select options and
8.            click Submit" runat="server" />
</form>
```

After you have included the control, it appears as displayed in Figure 5-2.


http://localhost/WebApplication1/WebForm1.aspx - Microsoft Internet Explorer

Address http://localhost/WebApplication1/WebForm1.aspx

Select date of birth :

<	September 2001					>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

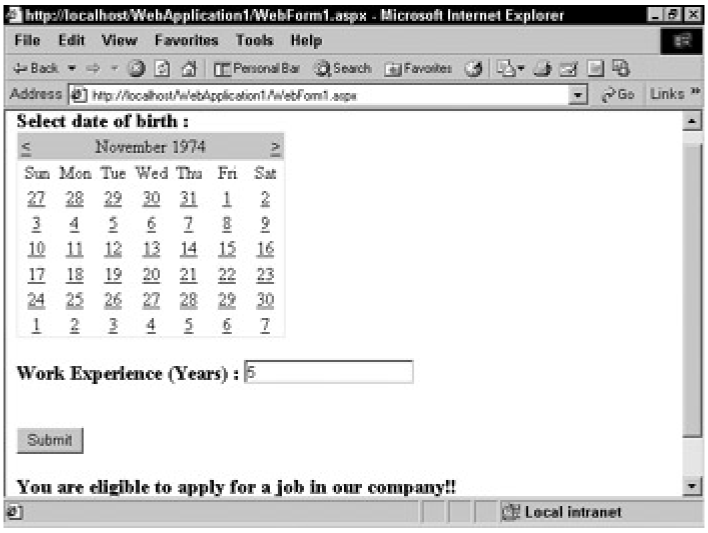
Work Experience (Years) : 0

Submit

Done — Local intranet

Figure 5-2: Composite control on a form

You can check whether the control is functioning correctly by selecting a date less than 1975 and specifying the work experience of more than five years. The output after you specify the aforesaid values is given in Figure 5-3.


http://localhost/WebApplication1/WebForm1.aspx - Microsoft Internet Explorer

Select date of birth :

<			November 1974			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

Work Experience (Years) : 5

Submit

You are eligible to apply for a job in our company!!

Figure 5-3: Data validation by the composite control

Adding Properties and Methods

After creating a custom control, you can add properties and methods to it and obtain the functionality you want in your control.

Properties are specifications that qualify the appearance and behavior of controls. In the case of standard controls, you can specify the property values either at design time or run time. At design time, the properties can be specified using the Toolbox. At run time, the implemented code sets the properties. If a property has a set accessor, the syntax to set a property would be as follows:

```
control.property = value
```

In this syntax:

- `control` signifies the name of the control.
- `property` signifies the property that you want to set for the control.
- `value` is the value that you specify for the property.

Methods are functions or procedures that provide a specific functionality to a control. Each Web Forms control has a set of methods associated with it. You can call a method of a control by using the following syntax:

```
control.method
```

In this syntax, control represents the name of the control and method represents the method associated with the control.

You can add properties and methods to custom controls also. To add a property to a user control, you need to write the code in the ASCX file of the control. Consider the example wherein you created a user control earlier in the chapter. To add properties, ProductID and ProductName, you need to add the following code in the class that implements the control:

```
<script language="C#" runat="server">


   public String ProductID
  {
   get
    {
     return Product_ID.Text;
    }
   set
    {
     Product_ID.Text = value;
    }
  }

   public String ProductName
  {
   get
    {
     return Product_Name.Text;
    }
   set
    {
     Product_Name.Text = value;
    }
  }
</script>
```

In this code, the get and set properties have been used. The get property is used to retrieve the value associated with the property. The set property is used to assign a value to a property**.**

Handling and Exposing Events

Each control has events associated with it. Events are generated as a result of user interaction or can be raised from other procedures. In Web Forms controls, the events are raised and handled on the server. An action is requested from the client side to the server with the help of a Web request. Then, the control raises an event on the server as a response to the client action. After the page or controls handle the event, the response is sent back to the client. This results in user experience similar to a desktop application.

Note Only the postback event can be posted to the server. User interface events that occur on the client side, such as mouse clicks or key presses, can only be communicated to the server by using postback events.

You can associate custom events with your controls. Handling user control events is more or less the same as handling events in any other Web Forms control. You need to decide whether to use the event handler in the containing Web Forms page or the user control. Writing event handlers in either of the cases is similar. However, you need to take some precautions if you decide to include the event handler in the user control. For example, if you have included the properties for the control in the existing Web Forms page, the properties will not be accessible from the user control unless you add functionality within the user control.

Let us create a button and add an event handler to it. Consider the user control that you created earlier in the chapter. You can add an event in such a way that whenever you write a value in the Product_ID text box and click a button labeled "Buy Now!," the name of the product will automatically appear in the Product_Name text box. To add the button to the control, add the following code:

```
<asp:Button id=Buy runat="server" Text="Buy Now!"
        OnClick="Buy_Click"></asp:Button>
```

The ID of the Button control is Buy. The button can achieve the desired functionality by the following code:

```
<script language="Vb" runat="server" ID=Script1>
  Sub Buy_Click (Src As Object, E As EventArgs)
    If MyProduct.ProductID="P001" Then
       MyProduct.ProductName="Toys"
    End If
  End Sub
</script>
```

This code is executed when a user clicks the button "Buy Now!" After a user clicks this button, the value of the TextBox labeled Product_Name is set to "Toys" if a user enters the product code as "P001."

Summary

In this chapter, you learned the basic structure of Web Forms controls and looked at the classes used for them. You explored the custom controls in detail. First, you learned how to create and use user controls. Then, you learned how to create and use composite controls. In this process, you learned how to use events, methods, and properties with custom controls.

Chapter 6: Validating User Input

Overview

This chapter covers the validation controls used in ASP.NET. These controls make page validation much easier and reduce the amount of code that the developer must write to perform page validation. The ASP.NET team reviewed numerous Web sites to determine the most common types of validation that were taking place on the Web. Most developers were reinventing the wheel to perform validation, so the ASP.NET team decided that Web developers needed a set of validation controls to add to their toolbox. From the start, these controls were designed to detect the version of the browser when used in client-side validation and then render the correct version of HTML for that client browser.

These research efforts lead to the development of the six controls covered in this chapter. The examples in this chapter will take a look at each control and explain the most commonly used properties for each control. However, keep in mind that all of the controls share basic properties, such as font, fore color, back color, and so on, so this chapter won't discuss those properties in detail. After you have read this chapter, you will have a firm understanding of validation controls and will be ready to use them in your own Web applications.

Understanding Validation Controls

Everything in the .NET Framework is a class, and the validation controls are no exception. All validation controls in the .NET Framework are derived from the BaseValidator class. This class serves as the base abstract class for the validation controls and provides the core implementation for all validation controls that derive from this class. Validation controls always perform validation checking on the server. Validation controls also have complete client-side implementation that allows browsers that support DHTML to perform validation on the client. Client-side validation enhances the validation scheme by checking user input as the user enters data. This allows errors

Part II: ASP.NET Database Programming

Chapter List

Chapter 8: Introducing ADO.NET
Chapter 9: Understanding Data Binding
Chapter 10: Working with Data Grids
Chapter 11: Using Templates
Chapter 12: Using SQL Server with ASP.NET
Chapter 13: Advanced Data Binding and XML

Chapter 8: Introducing ADO.NET

Overview

As more and more companies are coming up with *n*-tier client/server and Web-based database solutions, Microsoft with its Universal Data Access (UDA) model, offers high-performance access to diverse data and information sources on multiple platforms. Also, UDA provides an easy-to-use programming interface that works with practically any tool or language, leveraging the technical skills developers already have.

The Microsoft UDA model is a collection of Data Access Components, which are the key technologies that enable Universal Data Access. The Data Access Components include ActiveX Data Objects (ADO), Remote Data Service (RDS), formerly known as Advanced Data Connector (ADC), Object Linking and Embedding Database (OLE DB), and Open Database Connectivity (ODBC).

Microsoft is targeting many more such Data Access components that offer easy-to-maintain solutions to organizations. Such solutions are aimed at allowing organizations use their own choice of tools, applications, and data sources on the client, middle tier, or server. One of the emerging components within the UDAs collection is ADO.NET. This chapter introduces you to ADO.NET.

ADO.NET Basics

Microsoft ADO.NET is the latest improvement after ADO. ADO.NET provides platform interoperability and scalable data access. In the .NET Framework, data is transmitted in the Extensible Markup Language (XML) format. Therefore, any application that can read the XML format can process data. It is not necessary for the receiving component to be an ADO.NET component at all. The receiving component might be a Microsoft Visual Studio–based solution or any application running on any other platform.

Although ADO.NET preserves some of the primary concepts from previous ADO models, it has been chiefly stretched to provide access to structured data from diverse sources. ADO.NET provides access to diverse data sources by using a consistent and standardized programming model. ADO.NET is upgraded to offer several advantages over previous versions of ADO and over other data access components.

ADO.NET builds the foundation of data-aware .NET applications. ADO.NET brings together all the classes that allow data handling. Such classes represent data container objects that feature typical database capabilities — indexing, sorting, and views. While ADO.NET offers a solution for .NET database applications, it presents an overall structure that is not as database-centric as the ADO model.

The ADO model uses the concept of recordsets, which are the ADO representation of tables and views from a database. Although these recordsets are very flexible to use and allow access to data even when disconnected from data sources, they suffer from a major drawback. In the case of distributed and Web applications, data needs to be exchanged among different components at different tiers, which might be running on variety of platforms. Of course, the format of the data being exchanged should be understood by all components. This transmission of data requires the conversion of data types of values to some data types that are recognized by the receiving components. This conversion is called COM marshalling. Thus, the interoperability is limited when using ADO recordsets. So, the concept of ADO recordsets fails when we look at the Internet interoperability.

Like ADO, ADO.NET also allows you to access data when disconnected from actual data sources. However, unlike ADO, ADO.NET uses XML as the data format. Because XML is a universal data format being used, ADO.NET expands the boundaries of interoperability to the Internet. In addition, instead of recordsets, ADO.NET uses the DataSet and DataReader objects to access and manipulate data. You'll learn about these objects later in the chapter. Thus, ADO.NET is designed to perform better and be more flexible than ADO. However, to support ADO objects, the corresponding equivalents exist in ADO.NET.

Note Data container objects are the objects that contain data to be transmitted to the receiving components.

To take full advantage of ADO.NET, you should put some effort into understanding the concept itself, rather than simply figuring out the fastest way to port your code. Whatever .NET programming model you might choose — Windows Forms, Web Forms, or Web Services — ADO.NET will be there to help you with data access issues. The workings of ADO.NET are shown in Figure 8-1. Let us now look at the key features offered by ADO.NET.

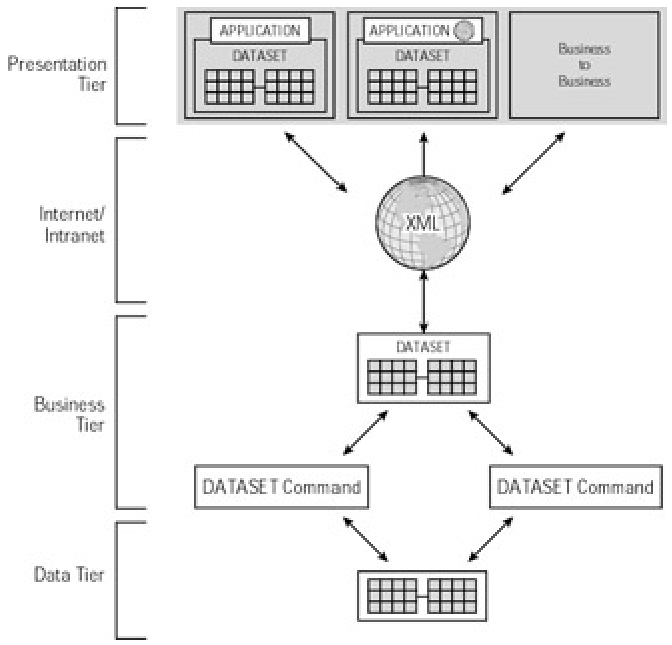


Figure 8-1: The workings of ADO.NET

Interoperability

The ADO.NET model is designed to take maximum advantage of the flexibility provided by the large industry acceptance of XML. ADO.NET uses XML for transmitting datasets among components and across tiers. Any component that is capable of reading the XML format can process the data. It is not necessary for the receiving component to be an ADO.NET component. The component that is sending or transmitting the dataset can simply transmit the dataset to its destination without bothering with how the receiving component is implemented. The component asking for the dataset, the destination component, can be implemented as a Visual Studio application or any other application. However, the important point to be considered is that the receiving component should be capable of accepting the XML file formatted as a dataset.

Maintainability

After an application is deployed, there might be a need for changes in the application. For example, the application might need substantial architectural changes to improve its performance. As the performance load on a deployed application server grows, system resources can become inadequate, resulting in higher response times. As a solution to these problems, the application might need to undergo architectural changes by adding tiers. Here, the problem is not the multitier application design, but rather the problem lies in increasing the number of tiers after an application is deployed. This transformation becomes easier if the original application is implemented in ADO.NET using datasets. In ADO.NET, the communication between tiers is relatively easy, because the tiers can transmit data through XML-formatted datasets.

Programmability
The ADO.NET model uses typed programming to manipulate objects. In *typed programming,* the programming environment or programming language itself recognizes the types of things that are important to users. To take full advantage of typed programming, you must know the things that are of interest to programmers and to end users. Consider the following code using typed programming in ADO.NET:

```
If TotalQty > DataSet1.ProductInfo("Baby Food").QtyAvailable
```

This code is equivalent to a line using non-typed programming and is easier to read by end users. An end user who has little or no programming experience can easily grasp the meaning of the condition being tested. Also, in non-typed programming, if the developer makes a spelling mistake by chance (for example, ProductInfo is spelled as ProdcutInfo), a run-time error will get generated. On the other hand, in typed datasets, errors in the syntax caused by misspellings are detected at compile time rather than at run time.

Performance

In ADO, while transmitting data across tiers using COM marshalling in the form of disconnected RecordSets, the values must be converted to data types that are recognized by COM. This results in poor performance. On the other hand, ADO.NET is designed to use disconnected data architecture, which in turn is easier to scale because it reduces the load on database (does not require any data type conversions). Thus, in the ADO.NET model, everything is handled at the client side, which in turn improves performance.

Scalability

The Web-based, data-centric applications require multiple users to access data simultaneously. This increases the demand on data to be accessed, making scalability one of the most critical features. Applications that use resources, such as database connections and database locks, cannot support more users to access data simultaneously, because eventually the user demand for the limited resources will exceed their supply. Because ADO.NET uses disconnected data access, applications do not retain database locks or active database connections for long durations. Hence, ADO.NET accommodates scalability by encouraging programmers to conserve limited resources, and allows more users to access data simultaneously.

ADO.NET Object Model

The .NET Framework is designed to change dramatically the developer's current style of developing applications, including the data access features. For the .NET applications, the primary data access technology to be used would be ADO.NET — the latest addition to the ADO model.

The ADO.NET Object Model is primarily divided into two levels:

- **Connected Layer:** Consists of the classes that comprise the Managed Providers
- **Disconnected Layer:** Is rooted in the DataSet

This section describes both the Managed Providers and the DataSet.

Managed Providers

Managed Providers are a collection of classes in the .NET Framework that provide a foundation for the ADO.NET programming model. The .NET Framework allows you to write language-neutral components, which can be called from any language, such as C++ or Visual Basic. In the .NET Framework, the OLE DB and ADO layers are merged into one layer. This results in high performance, and at the same time allows components to be called from any language. The Managed Data Providers include classes that can be used for the following:

- Accessing data from SQL Server 7.0 and later
- Accessing the other OLE DB providers

The Managed Provider for ADO.NET is the System.Data.OleDb namespace, which allows you to access OLE DB data sources. This namespace includes classes that are used to connect to OLE DB data sources and execute database queries to access and manipulate data. Some of the classes included in the namespace are described in Table 8-1.

Table 8-1: ADO.NET classes for OLE DB data sources

Class	Description
OleDbConnection	Represents an open connection to a data source.
OleDbCommand	Represents a SQL query to be executed against the data source.
OleDbDataReader	Corresponds to a forward-only, read-only RecordSet. It is a highly optimized and nonbuffering interface for getting the results of a query executed against the data source.
OleDbDataAdapter	Represents a set of data commands and a database

Table 8-1: ADO.NET classes for OLE DB data sources

Class	Description
	connection that are used to fill the DataSet and update the data source.
OleDbParameter	Represents a parameter that is passed with an OleDbCommand object.
OleDbError	Represents the errors that are generated by the data source.

The Managed Provider for the ADO.NET classes to access and manipulate data stored on a SQL Server is the System.Data.SqlClient namespace. Table 8-2 describes some of the classes in this namespace.

Table 8-2: ADO.NET classes for SQL Server

Class	Description
SqlConnection	Represents an open connection to a SQL Server data source.
SqlDataAdapter	Represents a set of data commands and a database connection to populate the ADO.NET DataSet object. The SQLDataAdapter class corresponds to the OleDbDataAdapter class.
SqlCommand	Represents a T-SQL statement or stored procedure that SQL Server will execute. The SqlCommand corresponds to the ADOCommand class.

Table 8-2: ADO.NET classes for SQL Server

Class	Description
SqlParameter	Used for passing parameters to the SqlCommand object. When the SqlParameter object of ADO.NET is used to pass a parameter to the SqlCommand object, the parameter represents a T-SQL statement or stored procedure. When SqlParameter object is used to pass a parameter to the SqlDataSetCommand object, the parameter represents a column from a result set. SqlParameter corresponds to the ADOParameter class.
SqlError	Collects information about run-time warnings and error conditions that an ADO.NET application encounters. SqlError corresponds to the ADOError class.

To demonstrate how to open a connection to a SQL Server database and fill the DataSet (discussed later in this section) with a database query result, consider the following code:

```
Dim connection As New
SqlConnection("server=localserver;uid=sa;pwd=;database=Sales")


Dim command As New SqlDataAdapter("SELECT * FROM Products Where
ProductID=@ID", connection)


Dim param1 As New SqlParameter("@ID", SqlDbType.Int)
```

```
param1.Value = 1

command.SelectCommand.Parameters.Add(param1)

Dim dataset As New DataSet()
command.Fill(dataset, "Products")
```

In this code:

- `connection` is a SqlConnection class object that represents a connection to the SQL Server database.
- `command` is a SqlDataAdapter class object that represents a set of data commands and a database connection.
- `param1` is a SqlParameter class object that represents the parameter to be passed in the T-SQL command.
- `dataset` is a DataSet class object that represents the DataSet that is filled by the query results.

DataSet class

The DataSet comprises the Disconnected Layer of ADO.NET. The DataSet consists of a local buffer of tables and relations. As shown in Figure 8-2, the DataSet object model consists of Tables, Columns, Relations, Constraints, and Rows. A DataSet contains a collection of DataTables (the Tables collection). A *DataTable* represents one table of in-memory data. A DataTable consists of the following:

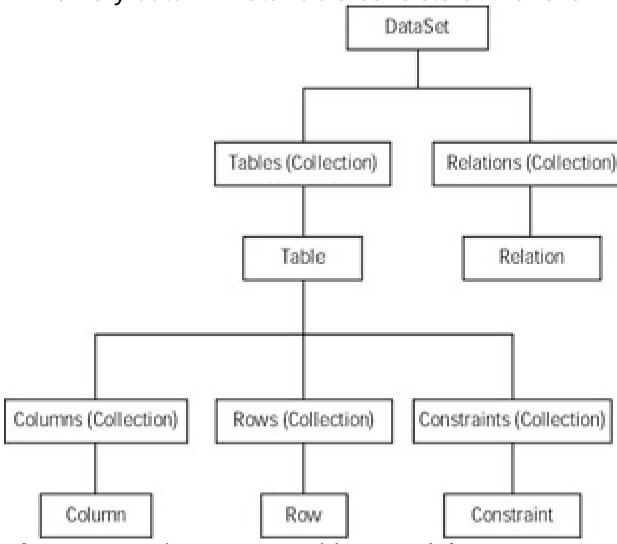


Figure 8-2: The DataSet object model

- A collection of columns (the Columns collection) that represents the table's schema.
- A collection of rows (the Rows collection) that represent the data held by the table.

A DataTable remembers the original state along with the current state, tracking the kinds of changes that have occurred. The data access classes are included in the System.Data namespace. Table 8-3 lists some of these classes along with their descriptions.

Table 8-3: The data access classes of ADO.NET

Class	Description
DataSet	Represents a complete collection of tables, relationships, and constraints. Both

Table 8-3: The data access classes of ADO.NET

Class	Description
	the System.Data.OleDb and the System.Data.SqlClient namespaces share this object of ADO.NET, making it the core component of the ADO.NET architecture.
DataAdapter	Represents a database query or stored procedure that is used to populate the DataSet object.
DataTable	Represents a data source that stores data in row and column format.
DataColumn	Represents a column in a DataTable.
DataRow	Represents a row in a DataTable.

Unlike RecordSets, which are equivalent to tables in ADO, DataSets keep track of the relationships between tables if any. The DataSet is designed with a rich programming model. The following code creates a new DataTable with the name ProductInfo:

```
Dim dset As DataSet = New DataSet("ProductInfo")
```

Later, you can add columns to the DataTable. The columns are added to the DataTable by using the Add method on the Columns collection, and the column is assigned a name and a datatype. Finally, data is added to the table by calling the NewRow method on the DataTable and storing the column values in each DataRow.

Changes from ADO

ADO.NET is an evolutionary improvement over ADO. Some of the improvements are described in Table 8-4.

Table 8-4: Feature changes in ADO.NET

Feature	ADO	ADO.NET
Memory-resident data representation	Uses the RecordSet object, which looks like a single table.	Uses the DataSet object, which contains one or more tables represented by DataTable objects.
Relationship between	Requires a JOIN query to combine data from	Provides the DataRelation object

Table 8-4: Feature changes in ADO.NET

Feature	ADO	ADO.NET
multiple tables	multiple tables in a single result table.	for combining data from multiple DataTable objects without requiring a JOIN query.
Data navigation	Scans the rows sequentially.	Uses a navigation model for nonsequential access to rows in a table. Tracks relationships to navigate from rows in one table to corresponding rows in another table.
Disconnected access	Supports the connected access, which is represented by the Connection object. To communicate with a database, ADO first makes a call to an OLE DB provider. However, ADO also supports the disconnected data access by the RecordSet object although it is not designed for it.	ADO.NET uses standardized calls for the DataSetCommand object to communicate with a database, which in turn communicates with the OLE DB provider. Sometimes, the DataSet object directly communicates with the APIs provided by a database management system.
Programmability	Uses the Connection object to transmit commands for mapping a data source that has an underlying data construct.	Uses the typed programming characteristic of XML. Data is self-describing because names for code items correspond to the "real-world" problem solved by the code. The underlying data constructs, such as tables and rows, do not appear, making code easier to read and write.
Sharing disconnected data between tiers or components	Uses COM marshalling to transmit a disconnected RecordSet. Only those data types that are defined by COM	Transmits a DataSet with an XML file. The XML format places no restrictions on data

Table 8-4: Feature changes in ADO.NET

Feature	ADO	ADO.NET
	standards support the feature to share disconnected data between tiers or components. Therefore, ADO needs to perform the data type conversions, which require system resources, resulting in low performance.	types. Therefore, ADO.NET requires no data type conversions, resulting in improved performance.
Transmitting data through firewalls	Problematic, because firewalls are typically configured to prevent system-level requests, such as COM marshalling.	Supported because ADO.NET DataSet objects use XML for representing data. Using HTTP as the transport, XML data can pass through firewalls.
Scalability	Database locks and active database connections for long durations result in limited database resources, allowing fewer users to access data simultaneously.	Disconnected access to database data without retaining database locks or active database connections for lengthy periods does not limit the database resources. This allows more users to access data simultaneously.

Communicating with OLEDB Data Sources Using ADO.NET

Every application that needs to retrieve data from a database needs to establish a connection to the database. In ADO, this was achieved using the Connection object. In ADO.NET, the classes to be used for establishing the connection depend on the data source being used. For instance, to connect to SQL Server databases, the classes in the System.Data.SqlClient namespace are used. To connect to OLE DB data sources, you need to use classes in the System.Data.OleDb namespace. It must be noted that both the SQL Server and OLE DB providers are managed providers. These providers act as a thin layer that connects the application to the database without adding any unnecessary overhead, such as converting from OLE DB-compatible data types to native SQL Server data types and vice versa when communicating between the client and the server. The SQL Server data provider, for example, does not depend on OLE DB/ODBC. Instead, it uses the Tabular Data Stream (TDS) protocol of SQL Server to natively communicate with the SQL Server. The use of the TDS provides a tremendous performance boost to applications.

Let us now look at the classes required to establish a connection to a Microsoft Access database using the OLE DB-managed data provider.

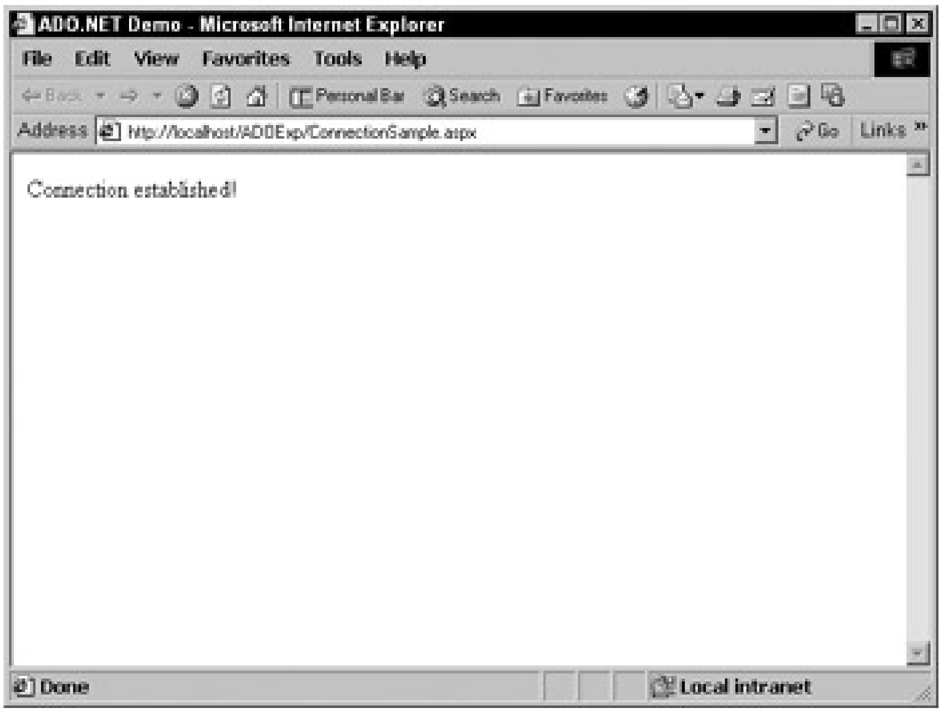
System.Data.OleDb.OleDbConnection class

This class encapsulates the connection to an OLE DB data source. Applications that need to use an OLE DB provider to connect to the data source should use this class, because not all data sources will have managed .NET data providers. When the .NET Framework is shipped by Microsoft, it provides managed data providers for some of the popular DBMSs like MS Access and SQL Server. But, the data providers for the other DBMSs will be developed by the respective vendors. To avoid waiting for the availability of managed data providers, ADO.NET has the option of connecting to any OLE DB-compliant data sources. The OLE DB provider makes it easier for the application to be migrated to ADO.NET. All the features of ADO.NET can be readily used in an application without having to depend on the availability of a managed .NET data provider.

Here is sample code that connects to an Access database:

```
<%@ Page Language="VB"%>
<%@ Import Namespace="System.data.oledb"%>
<html>
<head>
  <title> ADO.NET Demo </title>
</head>
<Script runat="server">
    Public Sub DBConnect()
      dim cnAccess as OleDbConnection
      'construct the OleDbConnection object
      cnAccess = new OleDbConnection("Provider=Microsoft.
Jet.OLEDB.4.0;Data Source=C:\ADODemo\Employee.mdb")
      'Open the database connection
      cnAccess.Open()
      Response.Write("Connection established!")
    End Sub
</Script>

<body>
<%DBConnect()%>
</body>
</html>
```

Figure 8-3 shows the output of the page.


ADO.NET Demo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Personal Bar Search Favorites
Address http://localhost/ADOExp/ConnectionSample.aspx
Go Links
Done
Local intranet

Connection established!

Figure 8-3: Output of the page demonstrating the usage of the OleDbConnection class

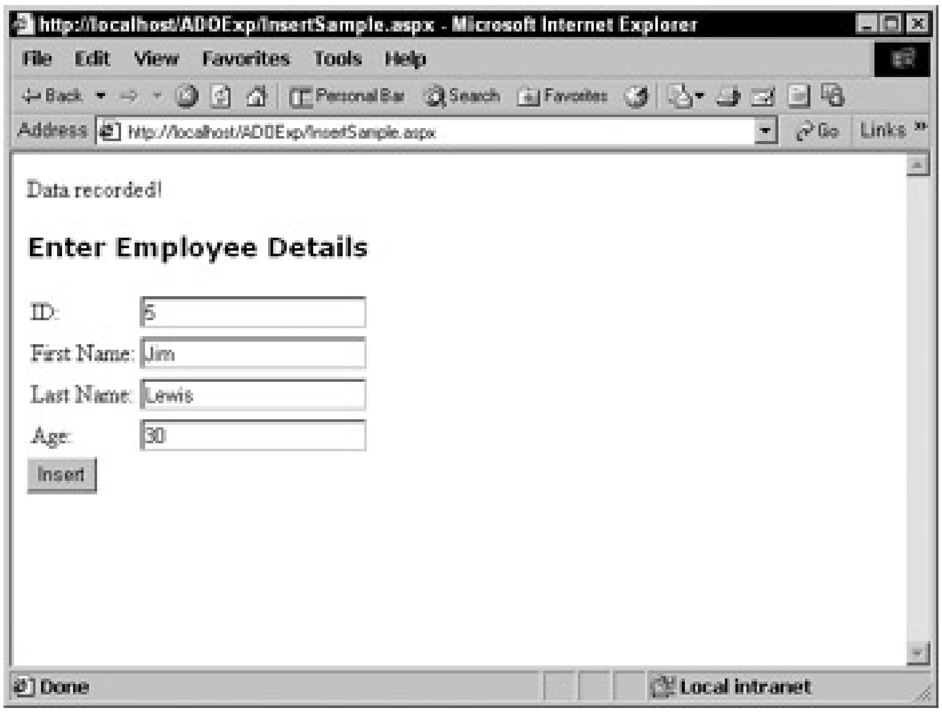
System.Data.OleDb.OleDbCommand class

This class encapsulates the commands that need to be sent to the OLE DB data source. Applications use the OleDbCommand class to create `select`, `insert`, `update`, and `delete` commands that need to be sent to the data source. Also, this class can be used to execute stored procedures besides sending input parameters to the stored procedure and retrieving output parameters from the stored procedure.

Here is sample code that inserts data using an insert command into an Access table:

```
<%@ Page Language="VB"%>
<%@ Import Namespace="System.Data.OleDb"%>
<html>
  <script language="VB" runat=server>
    Sub Insert_Click(Src As Object, E As EventArgs)
     ' Connect to Database
      dim cnAccess as New OleDbConnection("Provider=Microsoft.
Jet.OLEDB.4.0;Data Source=C:\ADODemo\Employee.mdb")
      cnAccess.Open()

      dim sID, sFName, sLName, sAge, sInsertSQL as string
      sID = eID.Text
      sFName = FName.Text
      sLName = LName.Text
      sAge = Age.Text

      'Make the insert statement
      sInsertSQL = "insert into employees values(" & sID & ",'" &
sFName & "','" & sLName & "'," & sAge & ")"
```

```
      'Make the OleDbCommand object
      dim cmdInsert as New OleDbCommand(sInsertSQL,cnAccess)

      ' This not a query so we do not expect any return data so use
      ' the ExecuteNonQuery method
       cmdInsert.ExecuteNonQuery()

      response.write ("Data recorded!")
    End Sub
  </script>

  <body>

    <form runat=server>
      <h3><font face="Verdana">Enter Employee Details</font></h3>
      <table>
        <tr>
          <td>ID:</td>
          <td><asp:textbox id="eID" runat="server"/></td>
        </tr>
        <tr>
          <td>First Name:</td>
          <td><asp:textbox id="FName" runat="server"/></td>
        </tr>
        <tr>
          <td>Last Name:</td>
          <td><asp:textbox id="LName" runat="server"/></td>
        </tr>
        <tr>
          <td>Age:</td>
          <td><asp:textbox id="Age" runat="server"/></td>
        </tr>
      </table>
      <asp:button text="Insert" OnClick="Insert_Click" runat=server/>
      <p>
      <asp:Label id="Msg" ForeColor="red" Font-Name="Verdana" Font-Size=
"10" runat=server />
    </form>
  </body>
</html>
```
Figure 8-4 shows the output of the page.


http://localhost/ADOExp/InsertSample.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/ADOExp/InsertSample.aspx
Done
Local intranet

Data recorded!

Enter Employee Details

ID:	5
First Name:	Jim
Last Name:	Lewis
Age:	30

Insert

Figure 8-4: Output of the page demonstrating the usage of the OleDbCommand class

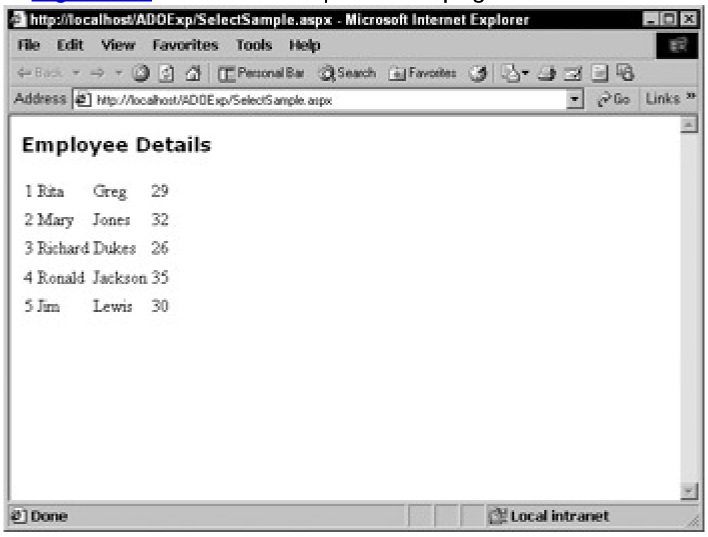
System.Data.OleDb.OleDbDataReader class

This class is equivalent to a forward-only, read-only Recordset object in classic ADO. This class is very useful to all applications that want to retrieve data returned from a query to the database and want to process one record at a time. A classic example for this would be to populate a list box with values retrieved from, say, a master table. The OleDbDataReader is independent of the OLE DB data source from which the data is retrieved. The process of reading data using the OleDbDataReader object is similar to reading data from any stream.

The following code retrieves a list of all the employees from an Access database:

```
<%@ Page Language="VB"%>
<%@ Import Namespace="System.Data.OleDb"%>
<html>
  <script language="VB" runat=server>
    Sub Page_Load()
      ' Connect to Database
      dim cnAccess as New OleDbConnection("Provider=Microsoft.Jet.
OLEDB.4.0;Data Source=C:\ADODemo\Employee.mdb")
      cnAccess.Open()
      dim sSelectSQL as string

      'Make the select statement
      sSelectSQL = "select * from employees"

      'Make the OleDbCommand object
       dim cmdSelect as New OleDbCommand(sSelectSQL,cnAccess)
```

```
        ' This query should return an OleDbDataReader so we use the
        ' ExecuteReader method
        dim drEmp as OleDbDataReader, sbResults as new StringBuilder()
        drEmp = cmdSelect.ExecuteReader()
        sbResults.Append ("<Table>")
        do while drEmp.Read()
         sbResults.Append ("<TR><TD>")
         sbResults.Append ( drEmp.GetInt32(0).ToString())
         sbResults.Append ("</TD><TD>")
         sbResults.Append ( drEmp.GetString(1))
         sbResults.Append ("</TD><TD>")
         sbResults.Append ( drEmp.GetString(2))
         sbResults.Append ("</TD><TD>")
         sbResults.Append ( drEmp.GetInt32(3).ToString())
         sbResults.Append ("</TD><TR>")
        loop
        sbResults.Append ("</Table>")
        lblResult.text = sbResults.ToString()
      End Sub
    </script>

    <body>
        <h3><font face="Verdana">Employee Details</font></h3>
        <p></p>
        <asp:label id="lblResult" runat="server" text=""/>
    </body>
  </html>
```
Figure 8-5 shows the output of the page.


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

Employee Details

```
1 Rita    Greg    29
2 Mary    Jones   32
3 Richard Dukes   26
4 Ronald  Jackson 35
5 Jim     Lewis   30
```

Figure 8-5: Output of the page demonstrating the usage of the OleDbDataReader class

System.Data.OleDb.OleDbDataAdapter class
The data adapter acts as the conduit between the client application and the database connection, command objects. The data adapter represents the command and connection that are used to populate the client dataset. In case of a disconnected client, the data adapter has the responsibility of firing the appropriate `insert`, `update`, or `delete` commands onto the database to synchronize the changes that are recorded in the client dataset.

The OleDbDataAdapter class has three command properties that are used to update the database:

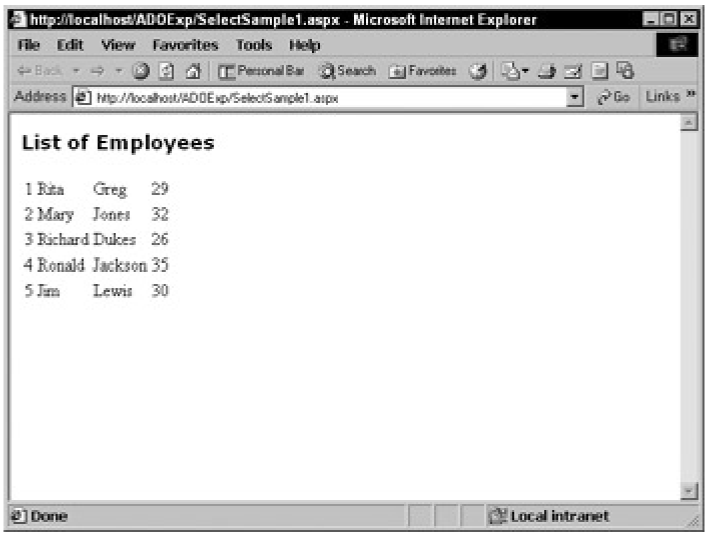
- `InsertCommand`: Represents the query or stored procedure that is used to insert new records into the data source.
- SelectCommand: Represents a SQL statement used to select records in the data source.
- DeleteCommand: Represents a SQL statement for deleting records from the data set.

System.Data.DataSet, System.Data.DataTable, System.Data.DataRow, and System.Data.DataColumn classes

The DataSet is a generic class provided by the .NET Framework. This class is very useful on the client side to store data in a manner that is much more functional and powerful than the ADO Recordset object. Moreover, the data in a DataSet is in XML format, and therefore is readily accessible and manageable. The XML format makes it very well suited to Web applications, and makes cross-platform access possible. The DataSet in memory is quite similar to a full-blown, in-memory DBMS in that it has the ability to store data from multiple tables and the relationships between them. The tables are stored in DataTable objects, and DataRelation objects represent the relationship between tables. The rows and columns in a table are stored in DataRow and DataColumn objects, respectively.

Let us look at an example that lists all the employees from an Access database by using a DataSet in a Web page:

```
<%@ Page Language="VB"%>
<%@ Import Namespace="System.Data"%>
<%@ Import Namespace="System.Data.OleDb"%>
<html>
  <script language="VB" runat=server>
    Sub Page_Load()
      ' Connect to Database
      dim cnAccess as New OleDbConnection("Provider=Microsoft.
Jet.OLEDB.4.0;Data Source=c:\ADODemo\Employee.mdb")
      cnAccess.Open()


      dim sSelectSQL as string
      ' Make the select statement
      sSelectSQL = "select * from employees"


      'Make the OleDbCommand object
      dim cmdSelect as New OleDbCommand(sSelectSQL,cnAccess)
      dim daEmp as new OleDbDataAdapter(cmdSelect)
      dim dsEmp as new DataSet
      dim sbResults as new StringBuilder()

      ' Fill the data with the output of the cmdSelect command. Note
      ' that the dataadapter is associated with the command. We use
      ' the dataadapter to fill the dataset.
      daEmp.Fill(dsEmp, "Employees")
      PrintRows(dsEmp)
    End Sub

    Sub PrintRows(ByVal myDataSet As DataSet)
       Dim dtEmp As DataTable
       Dim drEmp As DataRow
       Dim dcEmp As DataColumn, sbResult as new stringbuilder()
      ' Iterate through all the DataTables in the DataSet
       For Each dtEmp in myDataSet.Tables
        sbResult.Append("<Table>")
          ' Iterate through all the DataRows in the DataTable
          For Each drEmp In dtEmp.Rows
           sbResult.Append("<TR>")
             ' Iterate through all the DataColumns in the DataRow
             For Each dcEmp in dtEmp.Columns
              sbResult.Append("<TD>")
              sbResult.Append(drEmp(dcEmp))
              sbResult.Append("</TD>")
             Next dcEmp
          sbResult.Append("</TR>")
          Next drEmp
       sbResult.Append("</Table>")
       Next dtEmp

lblResult.Text = sbResult.ToString()
    End Sub

  </script>

  <body>
      <h3><font face="Verdana">List of Employees</font></h3>
      <p></p>
     <asp:label id="lblResult" runat="server" text=""/>
  </body>
</html>
```
Figure 8-6 shows the output of the page.


http://localhost/ADOExp/SelectSample1.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/ADOExp/SelectSample1.aspx
Done
Local intranet

List of Employees

```
1 Rita    Greg    29
2 Mary    Jones   32
3 Richard Dukes   26
4 Ronald  Jackson 35
5 Jim     Lewis   30
```

Figure 8-6: Output of the page demonstrating the usage of the DataSet, DataTable, DataRow, and DataColumn classes

Summary

This chapter introduced you to the new data access technology ADO.NET. First, you learned the basic features of ADO.NET. Then, you learned some of the most important ADO.NET objects. Here, you learned the objects to access OLE DB data sources as well as SQL data sources. To understand the advantages of ADO.NET over ADO, the features, such as disconnected data access, scalability, and interoperability of ADO and ADO.NET were compared. Finally, you learned to implement ADO.NET objects, such as OleDbConnection, OleDbCommand, OleDbDataReader, OleDbDataAdapter, and DataSet to access and manipulate data from OLE DB data sources.

Chapter 9: Understanding Data Binding

Overview

ASP.NET provides a rich set of controls that enable you to display information to users as well as accept information from users. You can display information in controls from a wide variety of data stores, such as properties, arrays, data structures, or databases. Some of the data stores are static, whereas others are dynamic. You usually use static data stores to display information for user reference. In addition to displaying static information in controls, there are situations that require you to display information dynamically. For example, you might need to display the discount based on the quantity purchased for a product. Also, you might need a control to display information from a database whose data keeps changing constantly. In such situations, ASP.NET provides a solution by providing a feature that allows data binding to controls.

This chapter introduces you to data binding with the ASP.NET server controls.

Introduction to Data Binding

Data binding means binding controls to information stored in a data store. Here, the term "data" is used in a very broad sense. When we talk about data binding, it implies binding any control property to almost any kind of data store. A data store can be as simple as a public property on a page, or as complex as a database stored on a server. This broad