1

Introduction to ASP.NET 2.0 and ADO.NET

Considering the speed and power that ASP.NET 2.0 and the .NET Framework 2.0 bring to the beginning programmer, it is a joy to write this edition. I believe you will find many moments when you just won't believe the quality of results you can achieve with little effort. I think you will often say, as I did at the first demos, "Wow, this is the way it should be." Hopefully, your managers and clients will be equally impressed by the speed and accuracy with which you produce Web sites.

This chapter presents an introduction to the topic of using data on an ASP.NET 2.0 Web page, how to set up a machine to use ASP.NET 2.0 with software and data for this book, and a set of initial demonstrations of the powerful features you will learn to use. Some of the material will review knowledge covered in the prerequisites (a basic knowledge of ASP.NET and familiarity with basic database tasks). The sections are organized as follows:

- Overview of the .NET Framework, ASP.NET, and ADO.NET
- Introduction to ASP.NET 2.0 and how ASP.NET 2.0 Web pages use data
- Guidance through the setup tasks required for the book
- Demonstration of several ASP.NET 2.0 pages that utilize data

In the middle of this chapter you will find a list of steps to follow to set up the software to do all of the exercises in this book, including the .NET Framework 2.0 (includes ASP.NET), an editor to create pages, database software, and some sample databases.

The last section walks you through three demonstrations, as follows:

Try It Out	Topic
1	GridView Table from XML with Paging and Sorting
2	DataList from Access
3	Tree View from XML

Overview of the Technologies

About a million developers have been creating Web sites using the .NET Framework in its first version. So in the summer of 2003, many ears perked up when rumors came out of Microsoft that a new version was available, which promised to decrease the number of lines of code required to create ASP.NET pages by 70 percent. An increase in productivity of that level does not come often in the world of programming. When samples of ASP.NET 2.0 code were demonstrated in the fall of 2003 at the Microsoft Professional Developer's Conference, the result was at least as good as the expectation. A Web page that warranted a budget of several hours of programmer time in the first version of ASP.NET could easily be built in a few minutes using ASP.NET 2.0. Simply stated, any programmer that continues to create ASP.NET pages in version 1 after the final release of the .NET Framework 2.0 is spending a lot of extra time to accomplish the same results.

Perhaps more than any other area, ASP.NET 2.0 offers advances in the ease of incorporating data into a page. Programmers no longer need to have detailed knowledge of connection, command, and data reader and data adapter objects to implement common data scenarios. ASP.NET 2.0 makes basic data use simple and brings more complex use of data within the grasp of beginners.

Introduction to the .NET Framework

Microsoft developed .NET as a philosophy and set of technologies for computers to work together in the world of the Internet. The overall objective was to provide a smooth flow of information and processes across a wide range of systems and devices. .NET is not a language or a specific product. Rather, it is a set of standards and guidelines that are incorporated into almost all Microsoft products released since about 2002.

.NET embraces a standardized format for the exchange of information using the open-standards XML format. Extensible Markup Language (XML) eliminates the need for a requestor to have any specialized knowledge about how the data store holds information — the data can always come out in the self-describing XML format. Likewise, almost all data stores now have the ability to serve up their information in XML, making them appealing to all .NET data consumers.

.NET supports the Web Services standard for software to request the running of code in remote software using the open-platform standards Simple Object Access Protocol (SOAP) and XML. A .NET Web site can find out from another Web site what services it offers and then consume those services. This makes it possible for a Web site to obtain HTML, calculated results, or sets of data from other Web sites.

As part of its .NET initiative, Microsoft released a runtime and set of programming tools and application program interfaces (APIs), called the .NET Framework, to enable the development community to build .NET-connected applications and XML Web Services. The .NET Framework is composed of the Common Language Runtime (CLR) and a unified set of class libraries.

The CLR provides a fully managed execution environment for running applications, providing several services such as assembly loading and unloading, process and memory management, security enforcement, and just-in-time compilation. What gives the Common Language Runtime its name is the ability to author applications in a wide variety of languages, and compile source code to an intermediate language that the CLR understands and can run regardless of the original source language. This "language independence" is a key feature of the CLR (and also of ASP.NET), and it allows developers to work in their preferred language, such as C#, VB, or Cobol, while still accessing common features in the .NET Framework.

The .NET Framework also includes a set of class libraries that provide common functionality that every application needs. These class libraries can be accessed from any language supported by the .NET Framework. The services (and corresponding namespaces) offered by these class libraries include the following:

- Base Types *(System)*
- Input/Output *(System.IO)*
- Data Access *(System.Data)*
- Security *(System.Security)*
- Data Structures *(System.Collections)*
- Configuration *(System.Configuration)*
- Networking *(System.Net)*
- Reflection *(System.Reflection)*
- Globalization *(System.Globalization)*
- Painting and Drawing *(System.Drawing)*
- Tracing and Diagnostics *(System.Diagnostics)*
- Windows (Client) Application Model *(System.Windows.Forms)*
- Web Application Model *(System.Web)*

Note that the .NET Framework contains two application programming models, one for client applications (System.Windows.Forms) and one for Web-based applications (System.Web). This book is concerned with the latter model. The System.Web namespace in the .NET Framework is the portion of the .NET Framework that provides ASP.NET functionality. In other words, ASP.NET is just one part of the overall .NET Framework for building applications.

Introduction to ASP.NET

ASP.NET is a programming model for building Web-based applications. It is essentially a runtime and set of .NET Framework class libraries that can be used to build dynamic Web pages. It runs within the context of a Web server, such as Microsoft Internet Information Server (IIS), and processes programming instructions on the server to service browser requests. Unlike static HTML, which is served directly from the Web server, ASP.NET pages are actually executed on the server to produce dynamic results. The final rendering of a page might be constructed from a variety of different instructions and/or data sources.

ASP.NET pages are stored under the .aspx extension. Pages are created by a programmer as a combination of text, markup (such as HTML), and ASP.NET server-specific tags and script, and then stored on the Web server. You can think of a stored ASP.NET page as a set of instructions for how to build an HTML page. When the page is requested, the server-side logic is processed to create a page in pure markup that the client browser can understand and render. Because the rendered output is pure markup, any browser can read it; all the dynamic processing happens on the Web server. ASP.NET server-specific tags are very powerful, including the capability to react to user actions, connect to data stores, and automatically build very complex HTML structures.

As previously mentioned, ASP.NET is simply part of the .NET Framework and, consequently, ASP.NET pages can take advantage of all of the services offered by that framework, including networking, data access, security, and much more. The fact that all of these framework services are available to ASP.NET enables you to build much richer Web applications than ever before. You can spend less time reinventing the basic building blocks that all applications need, and spend more time focusing on the specific logic that is unique to your application.

ASP.NET also introduces some unique innovations in Web programming that greatly improve the development model over classic Active Server Pages (ASP):

- **Language-independence** — Because ASP.NET is part of the .NET Framework, ASP.NET applications can be constructed in the language of your choice, for example C#, VB, or J#. Classic ASP, on the other hand, is limited to only JScript or VBScript pages.
- **Compiled instead of interpreted** — Unlike classic ASP, which interprets programming instructions every time the page is requested, ASP.NET dynamically compiles pages on the server into native programming instructions that can be run much, much faster. It is not uncommon to see orders of magnitude difference between the performance of a classic ASP page and the performance of an equivalent ASP.NET page.
- **Event-driven programming model** — In classic ASP, pages are always executed in a top-down linear fashion, and HTML markup is often mixed in with the programming instructions. Anyone with a moderate amount of experience in ASP knows that this can make your pages difficult to read, and even more difficult to maintain. ASP.NET introduces an event-driven model that allows you to separate code from markup content, and factor code into meaningful units for handling specific tasks, such as responding to a button click from the client. This VB-like eventing model greatly improves the readability and maintainability of your pages.
- **Server controls** — Classic ASP requires you to dynamically construct a page rendering by piecing together HTML fragments in code, which often results in writing the same code over and over again across your applications (how many times have you needed to construct a table from a database query?). One of the great advancements that ASP.NET brings to Web programming is the ability to encapsulate common rendering and behavior into *server controls* that can be

easily reused within an application. A server control is created declaratively, just like an HTML tag, but represents a programmable object on the server that can interact with your code and output a custom dynamic HTML rendering. ASP.NET includes 80+ server controls that encapsulate anything from standard form elements to complex controls such as grids and menus.

- **Design time improvements to controls (when used with Visual Web Developer)** — Developers can decrease the time it takes to develop a complex page by using design time interfaces such as smart task panels, tag-level navigation bar, and the wizards that can set control properties.

Introduction to ASP.NET 2.0

The first ASP.NET versions (1.0 and 1.1) rapidly spread throughout the Microsoft-centric developer community from 2001 to 2003. Programmers quickly appreciated that they could spend a lot less time programming using the power and flexibility of the .NET Framework, and CIOs saw that they could devote more resources to high-level improvements to their IT structure when programmers were spending less time troubleshooting custom code. ASP.NET was truly a monumental release that simplified the lives of developers.

However, even prior to the release of version 1, the Microsoft ASP.NET team was already working on ASP.NET 2.0. They set out with the following ambitious design goals:

- Remove 70 percent of the lines of code needed to build a typical Web application.
- Provide a set of extensible application services that provide the building blocks for common application scenarios such as membership, roles, personalization, and navigation.
- Create a rich set of scenario-based server controls that are able to leverage the aforementioned services to deliver complete and customizable user interface (UI) that exposes those services with a minimum of code.
- Improve the performance of IIS when it is serving pages in conjunction with the .NET Framework.
- Provide administration features that enhance the deployment, management, and operations of ASP.NET servers.
- Improve the tools for hosting companies to support multiple sites and to migrate developers' projects to public deployment.
- Enable nearly all features of ASP.NET to be easily extended or replaced with custom implementations for advanced scenarios.

It is worth pausing to reflect on that first goal, namely the removal of 70 percent of the code needed today to write a dynamic Web application. How is this possible? The ASP.NET team at Microsoft looked closely at the variety of common scenarios being implemented in custom code today, and specifically looked for ways to encapsulate those scenarios into building blocks, particularly server controls, that could accomplish those tasks automatically. For example, most Web applications need security or navigation, or personalization services to provide custom experiences for users. In ASP.NET 2.0 these scenarios are exposed as a set of configurable application services, and server controls that talk to those application services, to enable dramatic reductions in the amount of application code required to implement these common scenarios. Among all the common scenarios, however, one stood apart as absolutely

essential to every application, and that was data access. Data is the common thread that drives all dynamic Web applications, and so it is no surprise that the ASP.NET team defined some very aggressive goals toward reducing the amount of code and concepts necessary to perform data access in ASP.NET 2.0 applications:

- Enable a declarative (no-code) way to define a source of data in ASP.NET.
- Enable a declarative (no-code) way to display data in UI controls, without having to explicitly data-bind at the right time in the page execution life cycle.
- Enable a declarative (no-code) way to perform common data scenarios such as sorting, paging, filtering, updating, inserting, and deleting data.
- Enable a rich set of UI controls for displaying data, including flexible grid/details controls with the ability to both display and manipulate data.
- Enable an extensible model for building custom data sources to support new types of data.

ASP.NET 2.0 has a specific set of server controls for programmers to add data interactions to a page. The data-specific controls are divided into two groups: data source controls and data-bound controls. Data source controls create the link to the database. The data-bound controls take the information from the data source controls and create a rendering on the page. This simple two-control pattern is available in many variations. There are data source controls for many types of databases and even nonrelational data sources. Likewise, many data-bound controls render to page tables, trees, lists, and other formats of data. The preceding several pages provided an introduction to and theory behind ASP.NET, and we are now at the nuts and bolts of what is covered in the rest of this book: the use of data source controls and data-bound controls. Data source controls include the following, which ship with the product:

- SqlDataSource control to connect to the Microsoft SQL Server and other databases
- AccessDataSource control to connect to MDB files
- ObjectDataSource to connect to middle-tier objects
- XMLDataSource for XML files or data streams
- SiteMapDataSource for XML files in the format of the ASP.NET 2.0 site map

Additional controls are already in development by third parties.

Data-bound controls include many that are familiar from ASP.NET 1.x, as well as some that are completely new for ASP.NET 2.0:

- ListBox, DropDownList and BulletedList, CheckBoxList, RadioButtonList
- AdRotator to provide a data-bound control implementation of an old feature
- DataList and Repeater to provide data in flexible layouts
- DataGrid (same as ASP.NET 1.x) and GridView (new for version 2) for tabular data
- DetailsView and FormView to provide information for one record with easy navigation
- TreeView to display hierarchical data

Taken together, the data source controls and data-bound controls will represent the majority of effort in this book.

Prior to the public beta release of the .NET Framework 2.0, limited groups of programmers were shown working code to solicit early feedback. The uniform response was an enthusiastic "This is great!" followed by "How soon can I replace my ASP.NET 1.x applications with ASP.NET 2.0?" With the beta release now in hand, we need not wait any longer.

Introduction to ADO.NET

ADO.NET is a set of class libraries in the .NET Framework that makes it easier to use data in your applications. Microsoft has gathered the best practices in data connections from the past several decades and written the code to implement those practices. The code is wrapped up into several objects that can be easily used by other software.

The code within ADO.NET handles much of the plumbing and database-specific intricacies so that when ASP.NET page designers want to read or write data they can write fewer lines of code and code that is more standardized. ADO.NET, like ASP.NET, is not a language. It is a collection of objects (classes) that hold code written by Microsoft. You can run that code from outside the object using a programming language such as Visual Basic or C#.

You can think of ADO.NET as a very smart translation layer between the data source and a data consumer. ADO.NET can accept commands from the data consumer's language and turn them into commands that are appropriate to carry out the task in the data source. But, as you will see, ASP.NET 2.0 offers server-side data controls that make it even easier to work with ADO.NET, sometimes eliminating the need to use ADO.NET objects directly at all.

Many readers will already have experience with earlier versions of ASP.NET. This short section recalls that model for the purposes of demonstrating the ADO.NET objects you needed to work with to bring data into a Web page. For those readers who never used earlier versions, view this as a curiosity of history, akin to a study of surgery techniques prior to the discovery of ether. In the past, a typical simple ASP.NET version 1.x page required the following code:

```
<script runat="server">
  Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    BulletedList1.DataSource = GetAuthorsByState("CA")
    BulletedList1.DataBind()
  End Sub

  Shared Function GetAuthorsByState(ByVal state As String) As System.Data.DataSet
Dim connectionString As String =
"server=(local);database=pubs;trusted_connection=true"
Dim dbConnection As System.Data.IDbConnection = New
System.Data.SqlClient.SqlConnection(connectionString)

    Dim queryString As String =
"SELECT [authors].[au_id], [authors].[au_fname], [authors].[au_lname],
[authors].[state] FROM [authors] WHERE ([authors].[state] = @state)"
    Dim dbCommand As System.Data.IDbCommand = New System.Data.SqlClient.SqlCommand
    dbCommand.CommandText = queryString
    dbCommand.Connection = dbConnection

    Dim dbParam_state As System.Data.IDataParameter = New
System.Data.SqlClient.SqlParameter
    dbParam_state.ParameterName = "@state"
```

```
    dbParam_state.Value = state
    dbParam_state.DbType = System.Data.DbType.StringFixedLength
    dbCommand.Parameters.Add(dbParam_state)

    Dim dataAdapter As System.Data.IDbDataAdapter = New
System.Data.SqlClient.SqlDataAdapter
    dataAdapter.SelectCommand = dbCommand
    Dim dataSet As System.Data.DataSet = New System.Data.DataSet
    dataAdapter.Fill(dataSet)

    Return dataSet
  End Function
</script>
<html><head runat="server"><title>Untitled Page</title></head>
<body>
    <form id="form1" runat="server"><div>
      <asp:BulletedList ID="BulletedList1"
DataTextField="au_lname" Runat="server" />
    </div></form>
</body></html>
```

The preceding example executes a simple SQL SELECT statement against a database and binds the result to a bulleted list control. The page has a method named GetAuthorsByState that creates several ADO.NET objects to accomplish this task:

- A SqlConnection object represents the connection to the database server.
- A SqlCommand object represents the SQL SELECT command to execute.
- A SqlParameter object represents a value to be substituted for a marker in the command.
- A SqlDataAdapter represents the ability to fill a DataSet object from a command.
- A DataSet represents the command result, which may be bound to the BulletedList.

In the `Page_Load` event, the GetAuthorsByState method is called to retrieve the DataSet result and assign it to the BulletedList's `DataSource` property. We then call DataBind() to force the BulletedList to synchronize itself with the data result. The fact that we need to call DataBind() at the appropriate time in the page execution life cycle is a key step that ASP.NET 2.0 seeks to eliminate in the most common cases. In fact, in most cases, ASP.NET 2.0 completely eliminates the need to interact with ADO.NET. However, it is useful to understand the relationship between the aforementioned ADO.NET objects to discuss how ASP.NET 2.0 improves upon that model.

ASP.NET 2.0 and Data Access

ASP.NET 2.0 gives us an improved model for data access, which eliminates much, if not all, of the code that was required to perform data-binding in ASP.NET 1.x. First, you no longer have to programmatically instantiate, set properties of, and call methods of ADO objects as in the preceding listing. Instead, you can add simple server-side data controls to your page and set their attributes. When the page is rendered, ASP.NET 2.0 will automatically perform all of the object instantiation and method calls to set up and display your data. Compare the following listing in ASP.NET 2.0 to the last listing.

```
<html>
<head runat="server"><title>Demo</title></head>
<body>
    <form id="form1" runat="server">
        <asp:SqlDataSource ID="SqlDataSource1" Runat="server"
   SelectCommand="SELECT au_lname FROM authors WHERE (state = @state)"
   ConnectionString="Server=HPSERV;Integrated Security=True;Database=pubs">
            <SelectParameters>
                <asp:Parameter Type="String" DefaultValue="CA" Name="state" />
            </SelectParameters>
        </asp:SqlDataSource>

<asp:BulletedList ID="BulletedList1" Runat="server"
    DataSourceID="SqlDataSource1"
             DataTextField="Au_lname">
        </asp:BulletedList>
    </form>
</body></html>
```

The second improvement derives from the server-side controls being sensitive to events in the page life cycle. Appropriate actions occur at the right time by the ASP.NET 2.0 server-side controls. Notice that there is no reference in the ASP.NET 2.0 page to any event in the page life cycle. Students of earlier versions of ASP were typically confused by the intricacies of when in a page life to perform various tasks, particularly data-binding. Thus, many ASP.NET 1.x pages suffered from programmers writing code that either called DataBind under the wrong event, or called DataBind duplicate times in multiple events. These timings are now automatic with the ASP.NET 2.0 server-side data controls.

Note in the preceding ASP.NET 2.0 code that two server-side controls are used. The first is a data source control, in this case the SqlDataSource control. Behind the scenes the control sets up all of the ADO connection objects needed for the display of data, including the Connection, Command, and DataReader or Dataset objects. Then, a data-bound control named BulletedList is used to take the data of the data source control and actually render it to the page.

Review of Terminology

To round out the introductory sections, I'll provide a list of some of the terminology used so far and that will continue to be used throughout the book.

- **Dynamic Web Pages** — Files stored on the Web server as code and then converted to HTML at the moment they are requested. When they are converted they can take into account the real-time situation of the user and the owners of the Web site, and thus take different forms for different requests.
- **IIS (Internet Information Server)** — A built-in Web server in Windows that serves Web pages to requestors via TCP/IP. IIS operating on Windows 2000 or Windows XP Professional has the capability to use .NET Framework classes to serve ASP.NET Web pages.
- **.NET Framework** — A group of classes that hold code written by Microsoft to make applications easier and faster to develop and more suitable for operation over the Internet. Various classes are incorporated into a dozen or so Microsoft products that are .NET-enabled.

- **Common Language Runtime** — A feature of the .NET Framework that enables programmers to write in one of many languages, then compile the code to a single, uniform language for deployment.
- **ASP.NET** — A runtime and set of class libraries in the .NET Framework for building dynamic Web applications.
- **Data Store** — A place where data is kept and usually managed. All RDBMS are data stores, but some data stores are not RDBMS because they are not relational.
- **Database or Relational Database Management System (RDMS)** — The software that enables reading and manipulation of data. Most systems include tools to design and test databases as well as tools to optimize sets of procedures. An RDBMS must store data in compliance with some form of normalization (a relational format).
- **DataBase Schema (or Database Metadata)** — The structure of the database, including the design of the tables and relationships. Schema does not include the actual data values.
- **Microsoft™ Access** — An RDMS that is based on the MDB file format, the JET engine, and a series of tools for building and using databases. Access is inexpensive, easy to learn, widely understood, and currently deployed on many machines. However, it does not support more than a few concurrent users.
- **JET** — A database engine that runs in the background and uses MDB (Access) files. JET accepts commands directly from other software (such as .NET or Access) to read or modify MDB files.
- **Structured Query Language (SQL)** — A language used by data consumers to request a read or write from a data provider. For over a decade, SQL has been the standard for communication with a RDBMS.
- **Microsoft™ SQL Server** — An enterprise-strength RDBMS designed to support large amounts of data and many concurrent users.
- **Microsoft™ SQL Server Express (SSE)** — A freely available database engine based on the Microsoft SQL Server database engine. Unlike SQL Server, SSE is limited in the number of simultaneous data connections it can serve and has only a few utilities. This book uses SSE in most examples.
- **Microsoft™ Data Engine (MSDE)** — Similar to SSE, but based on an earlier version of the SQL Server engine. MSDE will work for the exercises in this book.
- **XML** — A standard format for data in which each value is stored and described. XML is not particularly efficient (the space required for the descriptions generally exceeds the size of the data), but it is easily read by many different data management systems.
- **Web Page Editor** — Software that allows pages to be opened and changed. The most basic editor is Notepad. Visual Studio, Visual Web Developer, and ASP.NET Web Matrix bundle an editor with other tools to improve productivity.
- **Integrated Development Environment (IDE)** — A set of tools that assists programmers in developing code. Visual Studio is a very powerful IDE; Web ASP.NET Web Matrix also offers many tools. A typical IDE includes a Web Page Editor.
- **ADO.NET** — A collection of classes (code) written by Microsoft that acts as an intermediary between data stores (such as Access or an XML file) and a data consumer (such as an ASP page).

- **Connection** — An ADO object that represents a unique path between a data consumer and data provider.
- **Command** — An ADO object that represents a SQL statement that can be passed to a database.
- **Parameter** — An ADO object that represents a variable piece of data that can be inserted into the Command Object (SQL statement) just prior to the statement going to the database.
- **DataSet** — An ADO object representing a group of data organized into records and fields.
- **Server Control** — A self-contained set of code (an object) that performs its tasks on the server to contribute to an HTML-compatible page that is sent to the browser. Server-side controls can maintain a sense of state through the ViewState.
- **Data Source Control** — A server-side control that creates a single, unique connection to a database. It provides an abstraction of the ADO objects and makes programming ASP.NET 2.0 pages faster and easier to build. Data source controls are available for Microsoft SQL Server Access, XML, and other sources of data.
- **Data-Bound Control** — A server-side control that takes data from a data source control and renders it on the page. Data-bound controls abstract from the programmer the HTML tags such as `<table>`. Data-bound controls are available to render tables, lists, trees, and other structures.

Setup Requirements for This Book

The installation wizards and packages are one of the last parts of the Beta 2 to be prepared. Therefore, I cannot be sure at the time this book goes to press of the exact steps for install. The following steps will be very close, but be prepared for a few changes. I will post to this book's downloads a section of notes on the final version of the install.

Keep in mind that this edition of the book is published with the release of a beta version of the .NET Framework 2.0. As with all beta software, stability and compatibility are not guaranteed. The safest way to work with beta software is to install it on a non-production machine, freshly formatted and with a newly installed OS. If you expect to do repeated installs, I suggest a machine with a slave drive of at least 10 GB. On the slave drive, make a copy of your install CDs for Windows, .NET Framework 2.0, drivers, Office, and any other software you may want to use. Also, store your user files (databases and ASPX pages) on the slave drive. With each new version of the framework, do a format of the master drive and reinstall the OS from the slave drive. Then install the beta software from the slave drive. Although this procedure takes a few hours (albeit unattended), it is the surest way to have a completely clean install for working with your beta software.

`www.wrox.com` provides all of the demonstration and exercise pages pre-typed for your use. Most of the examples in this book use the Northwind and Pubs databases for three reasons:

- Many readers have already encountered these databases and already have them on their server or hard drive.
- They are available in both Access and SQL versions, which works well for demonstration purposes in this book.
- They provide a wide variety of data types and relationships that you can use to explore several concepts in this book.

The "Install the Sample Databases" section later in this chapter provides installation instructions for both Northwind and Pubs.

We will start with an overview of what must be done to set up for this book in the sections below. Then we will walk through the actual steps back-to-back. The requisites follow:

Web server

.NET Framework 2.0

ASPX page editor

Database management software

Sample databases

Install a Web Server

Web pages must be processed by a Web server to be available to a browser. Two options work with the .NET Framework, one for deployment and one for development. Internet Information Server handles the load of a public Web site, but may also be used for development purposes. If you have already turned it on in your development machine, you will find that the .NET Framework automatically registers with IIS and is available for you to create ASP.NET pages. IIS is easy to enable in Windows. First, because it is a good habit to get into, check that you have updated your installation of Windows with the latest service packs and security patches. Then click through Start ➪ Control Panel ➪ Add & Remove Programs. Select Add/Remove Windows Components and add a check mark to IIS to install.

Microsoft also provides a lightweight alternative to IIS for developers, code-named Cassini, that is better suited to developers and students than IIS. The Cassini Web Server comes with Visual Studio and Visual Web Developer and installs automatically. When a page is run from the Visual Web Developer, Cassini will automatically start its Web service on a random port and invoke your browser with a request for your page sent to `http://localhost:xxxx/MySiteName/MyPageName.aspx`. Instead of IIS, Cassini will serve the page, including processing ASP.NET code and server controls. Evidence of the server activity appears as an icon system tray. Note that Cassini is designed for developers and does not scale adequately to support a publicly deployed Web.

For this book either Web server will work. Both servers use the same ASPX pages, so there is no need for any changes in syntax or commands. If you are already using IIS on your development machine, you will not have to take additional steps. If you have not set up and secured IIS, I suggest you use Cassini because it is automatically installed with Visual Studio or Visual Web Developer. Cassini will automatically download and install with VWD Express, so you will not see specific install steps below.

Install the .NET Framework Version 2.0

The .NET Framework version 2.0 is available for download in three ways. The first is the version named .NET Framework 2.0 Redist, which is about 20 MB. The second version includes the Software Development Kit (SDK) that includes the .NET Framework documentation, samples, and several SDK tools. However, the SDK download is significantly larger than the Redist, and this book does not rely on

features in the SDK. If you are not in a rush, you can download the .NET Framework and order the SDK on CD for a postage fee, as described on the download page. The third alternative is to use the version that is automatically downloaded and installed with Visual Studio or Visual Web Developer (VWD) Express. That route is both easiest and adequate for this book, and is described in the install of VWD Express that follows.

If you have the .NET Framework version 1 or 1.1 installed, you can upgrade to the .NET Framework 2.0 on the same machine. If you have already installed version 1.0 or 1.1 of the .NET Framework and either of those versions are already registered in IIS, the .NET Framework 2.0 setup will not *automatically* register 2.0 with IIS, since this would potentially upgrade applications to the new version without the user's consent. (This is a change in Beta2 from Beta1, which did in fact upgrade the server.) In order to register .NET Framework 2.0 if version 1.x is installed, run in a command prompt window `aspnet_regiis -i` from the .NET Framework 2.0 installation directory; e.g., `\WINDOWS\Microsoft .NET\Framework\<version>`. This command line utility will upgrade the server and all apps on it to use 2.0. Having said that and because this is beta software, I recommend that you install the version 2.0 of the .NET Framework on a clean machine — that is, without version 1.x of the .NET Framework.

Although it is possible to run both 1.x applications and 2.0 applications side by side on the same machine, the steps for doing so are more involved and outside the scope of this book.

Install an Editor to Create Web Pages

ASP.NET pages must be written using some type of editor. Most readers will select one of three options, depending on their budget and interest in learning to use new software. This book uses VWD Express.

Visual Studio

Visual Studio (VS) provides very powerful tools to design complex Web pages that employ multiple resources from throughout the enterprise. Objectives that require ten or twenty lines of typing in Notepad are performed in VS with a single drag-and-drop or by clicking through a wizard. After the drag-and-drop, VS will type all of the tags, attributes, and code to produce the feature. VS also provides intelligent assistance to typing so that you generally only have to type a character or two and VS will complete the syntax (called "IntelliSense" technology). Debugging features, including trace and immediate windows, are built into the product. For readers of this book, a key VS feature is the ability to read and display database information at design time so you do not need to have Access or a SQL tool open to see the structure and data of your database. Visual Studio includes many tools in multiple programming languages and tools for collaboration of multiple developers at a price starting (as of mid 2004) at around US $500. This book does not discuss the installation of Visual Studio.

Visual Web Developer Express

Visual Web Developer (VWD) Express is a new product from Microsoft that includes most features of Visual Studio needed for Web site development, but at a much lower price (not known at time of writing). All of the exercises in this text can be performed using VWD, as it includes Cassini, the visual design interface, all of the controls I discuss, and the debugging features. VWD does not include some of the large-scale deployment features of the Visual Studio used by teams of developers, but for the purposes of this book you will not need them.

Notepad and Other Editors

Notepad is free with Windows and can be used as a no-frills text editor to create ASP.NET 2.0. Remember that ASPX files are saved as ASCII text files, the same as HTML. Therefore, every exercise in this book can be typed into a Notepad file, saved with an aspx extension (not .txt), and will run fine. On the other hand, Notepad does not offer much assistance other than cut, copy, and paste. If you use Notepad, be prepared to do a lot of typing, look up a lot of syntax, and perform a lot of troubleshooting.

Web ASP.NET Web Matrix is an open-source project that offers a development environment for ASP.NET version 1.x pages. The application is available for free from `www.ASP.NET`. Installation is not difficult, and if you have used Visual Studio or InterDev, the interface looks very familiar. ASP.NET Web Matrix offers clickable property settings, an object browser that allows you to look up syntax, and color-coding. As of this writing, ASP.NET Web Matrix does not support the features of .NET version 2. Therefore, at this point it is ill-suited for the examples of this book.

Because readers will be using various editors, I try to accommodate all. Initial discussions are oriented towards Visual Web Developer because it offers strong drag-and-drop features and wizards at a low price. The resulting code is then presented so typists can enter the page as discussed into Notepad or another editor. All of the files are also available for download from `www.wrox.com`.

Install a Database Management System

This book requires a way to manage data for examples. Several choices are listed here; of those I recommend and use SQL Server Express (SSE) in this book.

- **Microsoft Access** is familiar and already widely deployed, but not recommended for use in a production Web site. It works for learning and development purposes, however.
- **Microsoft SQL Server** is a powerful choice for public deployment but is expensive and more difficult to set up and manage on a development machine.
- **SQL Server Express (SSE)** is a lightweight and free database engine based on the SQL Server Yukon engine (currently in beta release). Because SSE has the same syntax as SQL Server but lower cost and complexity, it greatly simplifies working with local data in a Web application. SSE is used to support this book's examples in most cases.
- **Microsoft Data Engine (MSDE)** is a freely available lightweight version of SQL Server. It is easy to install and stores data in a format that can be directly ported to a full SQL Server installation at time of deployment.
- **Other Relational Databases** such as Oracle or MySQL can be used. However, this book does not discuss their installation or management.

Another option, XML, has emerged as an important format for storing and transferring data. XML and other hierarchical data are addressed in a separate chapter at the end of this book. Other data formats, such as Excel and flat files, can also be used in special cases. They are not covered in this edition of the book.

Microsoft Access

Microsoft Access is sold as part of Microsoft Office Professional or as a separate purchase. When you install Access, you automatically get an install of a sample database named `Northwind.mdb` and the JET database engine. You can also download the `Northwind.mdb` file from the following:

```
www.Microsoft.com/downloads/details.aspx?FamilyID=c6661372-8dbe-422b-8676-
c632d66c529c&DisplayLang=en
```

There are two major problems with using Access as a source of data for Web sites, even in development environments:

- Access was not designed to scale well to significant loads. When loads begin to increase Access uses large amounts of server resources. Therefore, it is unsuitable for deployment in scenarios with more than five or ten users.
- The second problem concerns the syntax of passing information to Access when modifying data. Access relies on a model where the order of the data passed to a parameterized command determines the fields where the data is stored. This creates a number of problems for the page designer, as you will see in later chapters.

Microsoft SQL Server, MSDE, and SQL Server Express use a different model where the parameter values are named; thus, order is unimportant.

For the reasons just described, this book recommends that you not use Access for working with data in ASP.NET; instead, download the SQL Server Express (SSE) database engine described shortly.

Microsoft SQL Server

Microsoft SQL Server is an enterprise-level database management system designed to scale and perform for the needs of entire organizations. It makes an ideal RDMS for public Web sites with high volumes and the need for integration with all other Microsoft products.

Microsoft, to this point, has offered time-limited trial versions of the product at `www.Microsoft.com/sql`. SQL is generally installed by a database administrator, and the steps are outside the scope of this book. If you are installing it yourself, I suggest you select the mixed authentication security model and be sure to install the sample databases. If you are using an existing SQL server installation at your workplace, you may have to ask your administrator to reinstall the Northwind and Pubs sample databases. You will also need to know the name of the server, instance, user ID, and password.

Other Relational Databases

.NET can use other sources of data, including MySQL, Oracle, and any other database for which a provider exists. Those connections are covered in Chapter 4. In theory, you could build Northwind in one of those systems for the purpose of writing the exercises in this text. This book does not cover installation or maintenance of these other sources.

SQL Server Express

The SQL Server Express (SSE) edition provides an engine that holds a database and responds to SQL statement requests. SSE is built from Microsoft SQL Server (in fact, the next generation) and thus

responds exactly like the full-scale SQL Server. SSE is the next generation to the Microsoft Data Engine (MSDE), and the upgrade path to SQL Server is seamless. As of this writing, it does not include graphical tools for managing the database, changing the schema, or modifying data, although you will be able to use Visual Studio or Visual Web Developer to perform some of these tasks. I expect to see a basic GUI interface added (perhaps even with the Beta 2), named "Express Manager or XM Tools."

There are several advantages to SSE over other sources of data for readers of this book:

- It is free and installs automatically with VWD
- A developer spends little time acclimating to SSE — it operates as a background service without a visual interface to learn.
- The data format and organization is the same as Microsoft SQL Server, so when you are ready to deploy there are very easy and well-documented solutions. Many Web-hosting companies have extensive experience bringing SSE databases from developers up to SQL Server.

The biggest disadvantage is that in the Beta 2 timeframe, there is no GUI interface for installing or importing a new database. I talk you through the steps of installing your sample databases from the command line.

Setup

Having covered the software needed for running, creating, and testing data-driven ASPX pages, it is time to walk you through the setup. As I mentioned earlier, the final install wizards may be changed at the last minute, so check your download for additional notes. Although you probably will not have problems installing the Beta 2 with the Beta 1 installed, you can be more certain of success by taking the clean-disk approach described in the beginning of the chapter. All elements of .NET 2.0 will harmoniously co-exist with .NET 1.1.

Obtain the download of files specific for this book

First go to `www.wrox.com` and get the download for this book. It will contain notes on install, the sample database setup files, demonstration pages used in the text, and all of the exercises. Read the install notes.

Obtain and install Visual Web Developer (which installs SSE and Cassini)

Visual Web Developer is a one-stop package for most of what is needed in this book. Its wizard will install all of the Microsoft-provided components discussed in the last section: the .NET Framework 2.0, the Visual Web Developer development tools, the Cassini Web server and SQL Server Express.

1. Shut down all applications.
2. Visit the Developer Express page at `http://lab.msdn.microsoft.com/express` and follow the instructions for downloading and installing Visual Web Developer Express. While on that page, check for any notices regarding best practices, security, and changes to the install procedure.
3. Select the Visual Web Developer Express download and run it.

4. When and if prompted, you need to install VWD, SSE, and the Framework. The help library is recommended, and the SDK is optional (it provides sample code and FAQs). If there is a set of quick-start sample applications available, they can be a useful reference. This book does not use the J# component.
5. VWD will give you the option to install SSE. Agree, and (if asked about features) select SQL Server Database Services, SQL Command Line Tools, Connectivity Components, and the Management Tools. The SSE installation may proceed in the background. The default authentication mode will be Windows Authentication. If the install pauses and asks you to specify the authentication, then select Windows Authentication.
6. Finish the install and reboot.

After this step, you can perform two confirmations. Your Start ➪ Programs will now contain Visual Web Developer 2005 Express Edition Beta. Your Start ➪ Control Panel ➪ Administrative Tools ➪ Services will display SQL Server running. Note that your SSE instance will be named (local)\SQLExpress. You can now use SSE as your database management system, but keep in mind that the engine does not yet hold a database

Install the Sample Databases

Install the databases used in this book into SSE by using the provided scripts of SQL statements. These scripts hold several thousand lines of SQL statements that create the entire structure of tables, relationships, queries, and data for Northwind and Pubs.

1. Look within the data folder of the download for this book to find `instPubs.sql` and `instNwind.sql`. Copy the files to `C:\Program Files\Microsoft SQl Server\90\Tools\binn`.
2. Start a command prompt and navigate to the folder mentioned in step 1.
3. Run the following command (do not change the case of any characters). If you are curious about osql, you can see the parameter dictionary with the command `osql -?`.

   ```
   osql -S(local)\SQLExpress -E -i"instNWIND.SQL"
   ```

   You will see thousands of SQL statements scroll by as they execute to create the Northwind database and fill it with data.
4. Repeat for the Pubs database, again making sure not to change case.

   ```
   osql -S(local)\SQLExpress -E -i"instpubs.SQL"
   ```

5. Try testing your installation with the following commands:

   ```
   osql -S(local)\SQLExpress -E -d Northwind -q "SELECT FirstName FROM Employees"
   ```

6. Type **Quit** and press Enter to end the test. Test Pubs as follows:

   ```
   osql -S(local)\SQLExpress -E -d Pubs -q "SELECT Au_LName FROM Authors"
   ```

7. Type **Quit** and press Enter to end the test, then close the command prompt.

Connect VWD to our sample databases

We'll finish our database setup by making the two exercise databases easily available for our pages. VS and VWD can connect to a database and test the connection before using the data on a page. Any database is available, but the following steps give you some GUI management tools from within VWD. Once connected, these IDEs enable some exploration of the database and drag-and-drop creation of data controls on pages.

1. Start VWD and, in the menu, click through View ➪ Data Explorer (or Server Explorer in Visual Studio).
2. Right-click on DataConnections and select Add Connection.
3. On the left panel, click on Providers and select .NET Framework Data Provider for SQL Server.
4. On the left panel, click on Connection. Drop your list of servers and select local or the name of your machine or an alternate server name that you are using.
5. Accept the default logon using Windows NT security and then select a name, in this case, Northwind. Because we are using mixed authentication, check the "save Password" in this dialog box.
6. Click on Test Connection to get a confirmation and then OK to close.
7. Repeat for the Pubs database that you installed.

Note that your Data Explorer (Server Explorer) now has a connection entry for Northwind and Pubs. You can expand the entry to see the tables, and when you right-click on a table you will be offered an option to Show Table Data. You can demonstrate the power of this view by changing some small value, such as the spelling of a person's name. After a database is installed to MSDE using the osql at the command line, you can use VWD as a front end to examine a database.

Demonstrations

To finish off the first chapter, we will create several pages that demonstrate using ASP.NET 2.0 pages with data. I will talk you through creating the pages using the VWD tools. In addition, each of these pages is available in this book's download at `www.wrox.com`. Also, check at that location for the latest updates and warnings to match these exercises to the release version of the beta software.

Try It Out #1 — GridView Table from SQL with Paging and Sorting

In this exercise, we will quickly build a page that uses a GridView control to produce a table of author names that is sortable and supports paging. Then we will add a clickable filtering process to show authors living in just one state. We will finish with a link to a second page that shows more details about one of the authors.

1. Start VWD and Click through File ➪ NewWebSite ➪ General (Other languages/VisualBasic) ASP.NET Web site. Name the site `C:\websites\BegAspNet2Db`.
2. In the solution explorer (use View menu to turn on), right-click on the name of the site and click on New Folder; name it `ch01`.

3. Right-click on ch01 and add a new item of template type Web Form; name it `TIO-1-GridViewSQL`. Set language to Visual Basic and set "Code in Separate Page" to off. Leave other settings at their defaults (select master page = off). Once the nascent page appears on the VWD screen you will see, at the bottom left, tabs for Design and Source views. Switch to Design view.

4. Click through Menu:View ➪ Database Explorer (Server Explorer) to display the explorer of data connections.

5. Expand the Pubs connection, Tables folder, and the Authors table. Using Control+click, select the four fields au_id, au_lname, au_fname, and state. Drag them onto the page. VWD will automatically create a GridView and data source control for you.

6. The Tasks menu automatically opens next to the new control (or open it by selecting the GridView and clicking on the small arrow at the top right of the control). Enable paging and sorting, then click on AutoFormat and select one that you like. Click Apply. Save the page with Ctrl+S and press Ctrl+F5 to run it. When warned about debugging being disabled, select to modify the Web.config file to enable debugging. In ASP.NET 2.0, the first viewing of a page is delayed as the framework compiles and optimizes the code. Subsequent user hits are much faster.

> **Depending on your security settings, you may get a firewall blocked warning; if so, unblock the firewall for the** `WebDeveloper.WebExpress.exe`**.**

7. Play with the sorting (click header of each column) and paging (numbers at bottom of table). Close the browser so the page is unlocked for improvements back in VWD. Note that we have used drag-and-drop, but no code, to create a sophisticated display of data on the page.

8. As an alternate way to show data using two steps, go back to VWD and stay in Design view. Press Ctrl+Home, press Enter twice, and then Ctrl+Home again to get the insertion bar above the GridView and two lines of space for some working room. You will now add a display of data where you can be more specific about the source. You do this by adding the data source control separately from the data-bound control that displays the data.

9. Display the toolbar by clicking on Menu:View ➪ Toolbox. If the sections of the toolbox are collapsed (you see only the dark gray headings of Standard, Data, Validation. . .), expand the Standard and Data sections. Drag a SqlDataSource control to the top of the page.

10. The smart tasks panel will automatically open; click on Configure Data Source. Click on New connection, for server name enter `(local)\SQLExpress`, and use Windows NT integrated security. Drop the list of databases and select Pubs. Test the connection and then finish with a click on OK. Back in the Configure Data Source dialog, click Next. Do not save this connection string to the application configuration file because you want the string to be more obvious in this first demonstration. Select the table named Authors and click on just one column from the check boxes: State. Turn on Return only unique rows, and click Next. Test the query and then click Finish. You now have a SqlDataSource control named SqlDataSource2, the first step in this two-step method to add data to the page.

11. Display the list of distinct states by dragging a list box from the Standard section of the toolbox to the top of the page. In the list box's smart task panel, click on Choose Data Source and select SqlDataSource2 (the distinct states list). The display and value fields can both remain State. Click OK. Back on the smart task panel, click on Enable AutoPostBack and close the task menu, save, and press Ctrl+F5 to view in your browser. You now have a list control on the page along with an invisible SqlDataSource control. Close the browser before returning to VWD.

12. This step links the list box to the GridView, so only authors from the selected state appear. Select the SqlDataSource1 control (below the GridView — be careful you don't use the #2 control) and open its CommonTasks menu. Click on Configure Data Source, keep the same connection (AppConnectionString1), and click Next. If needed, change the option button to specify columns and check off the same four fields (id, firstname, lastname, and state). Now click on the WHERE button. In this box, you specify which authors to show. You want only those whose State value equals the selection in the States list box. Under the Column list, select State. The operator can stay as equals, and the source should be set to Control. Note the new options to pick a Control ID; you want ListBox1. Set the default value by typing in the string `CA` to start with a view of just authors in California. Click the Add button to append that `WHERE` clause to your SQL statement and then click OK, and Next. In the next page of the wizard, test the query using the default value of CA. Click Finish and then close the CommonTasks menu. If you are asked to refresh the control, click on Yes, save and hit Ctrl+F5 to run. Select various states and note the change in the table. Close the browser when you are done.

13. Now you will create a second page to display details about one author from our GridView. Check that you exited your browser. In VWD, click your Solution explorer tab so it is on top. Right-click on ch01 and add a new item of template type Web Form and name it `TIO-1-GridViewSQL-Details.aspx`. At the bottom left of the screen, select the Design view. Drag and drop a DetailsView control from the toolbar (Data section). Choose a New data source and select the type of (SQL) database. Choose the connection named AppConnectionString1 and click Next. Check on the Specify columns, click on the asterisk (all columns), and click Next. Click on the WHERE button and set the au_id column equal to a source of `querystring` named `au_id`. Set a default value of `213-46-8915`. Click Add and double-check that the SQL expression shows `[au-id]=@au_id`. Click OK, clink Next, test the query, and click Finish. Close the Details view control's CommonTasks menu, save, and press Ctrl+F5 to run. You will only see the details on one author.

14. Now that you have a page to show details about an author, you can finish by giving the user a way to click on the GridView to jump to the details page. Check that you exited your browser; then, in VWB, open `TIO-1-GridViewSql.aspx` in Design view. Select the GridView and expand the little top-right arrow to see the CommonTasks menu. Select Edit columns to see the Fields dialog box. Select Hyperlink in the Available Fields and click Add. Keeping your hyperlink field selected in the lower left, go to the properties on the right and set the following:

```
Text =  ViewDetails...
DataNavigateUrlFields =  au_id
DataNavigateUrlFormatString =  TIO-1-GridViewSQL-Details.aspx?au_id{0}
```

    Click OK and close the smart task panel.

15. Save and run. Observe the behavior in the browser. If you are typing outside of Visual Studio or VWD, the finished pages should appear as shown in Figure 1-1:

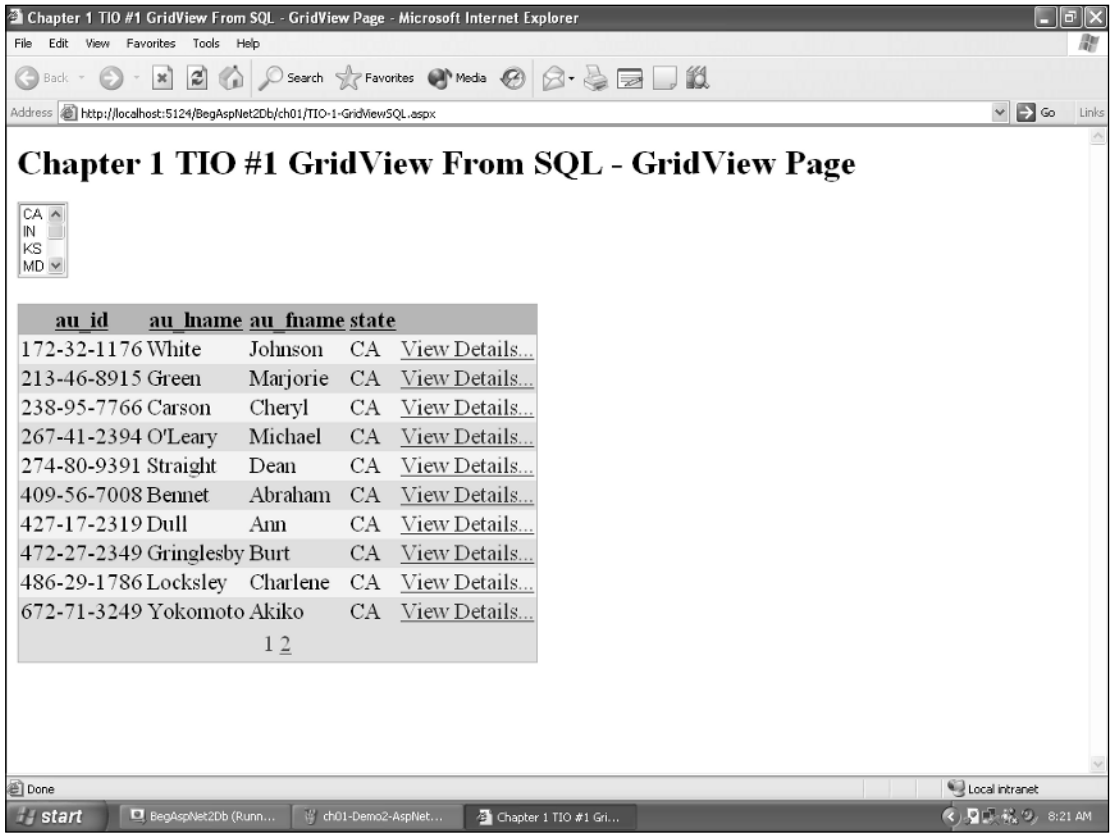

Figure 1-1

How It Works #1 — GridView Table from SQL with Paging and Sorting

This exercise covered a lot of small steps. When finished, you have two pages of medium complexity (similar to the type that will be discussed in detail in the middle chapters of this book). For now, there are five general concepts you should observe:

- All of the steps were performed in the VWD designer interface. You did not write any Visual Basic or C# code, nor did you directly write any HTML tags or set tag attributes.
- Take a look at the code that VWD generated for you by clicking on the Source view of the file `TIO-1-GridViewSQL` (or looking at the preceding code listing). Note that there is nothing in the initial `<script>` tags. All of the characters on the page are in standard HTML syntax with opening and closing tags, hierarchical tags, and attributes within tags.
- Look at the general pattern of the pages. There are pairs of controls: data source controls (SqlDataSource) and data-bound controls (GridView, DetailsView, and Listbox). The first gives you a connection to a database. The second displays data. For example, at the top of the GridView page, you have an `<asp:SqlDataSource>` and then an `<asp:ListBox>` that uses the SqlDataSource as its DataSourceID. You will study this pattern in detail throughout the book.

- Notice how a data source control can have its attributes controlled by another control on the page. The SqlDataSource for the GridView has a `WHERE` clause that is modified by the value the user selects in the list box. The information is passed in the form of @state in a tag called `<SelectParameters>`.
- You can pass a parameter from one page to the next in the querystring, and that parameter can be used to modify the DataSource on the new page. For example, in the `HyperlinkField` tag of the GridView you specified to pass the au_id field. The DataNavigateUrlFormatString added the target page's text to the au_id value. Then on the Details page, you set your SqlDataSource SelectCommand to use the query string value.

Overall, even on a first try, in less than 15 minutes you can produce pages with functionality that would have taken many hours in earlier versions of ASP.

Try It Out #2 — DataList from Access

In this exercise you will use a DataList control to display data with flexibility to arrange your fields within a template space. For variety, use an Access data source.

1. Double-check that a copy of `Northwinds.mdb` is in a folder named `App_Data` of your Web site.
2. Right-click on CH01 and add a Web Form item with the name `TIO-2-DataListAccess.aspx`.
3. In Design view, drag a DataList control from the Data section of the Toolbox to the page. Choose a data source = New Data Source. Select the data source type of Access database and accept the default name of AccessDataSource1. Choose a database file by browsing to `C:\websites\BegAspNet2Db\App_Data\Northwind.mdb`. Click Next and, for the Name (which means of a table or query), select Categories and check the ID, Name, and Description fields. Click Next and run the test query. Then click Finish. You now have a DataList control that points to categories of Northwind and in the background creates an AccessDataSource control.
4. Select the DetailsList, open its smart task panel (small arrow at top right) and click on the property builder to set some overall properties for the details list. Change the columns to 3 in the `RepeatColumns` property and the direction to horizontal for Repeat Direction. Click Apply and OK.
5. Now you will configure the display of each record's data. Continuing with the DataList selected, in its smart task panel click on Edit Templates. You can now add and format controls within the white ItemTemplate space. First select and delete the six controls in the template. Check that your insertion bar is in the editable space, then type the word **Item**. Now click through Menu:Layout ➪ Insert Table. Set the table to be 2 rows by 2 columns with a border of 2, and set the Cell Properties Background Color to a light shade of yellow. Click OK to create the table in the DataList's template.
6. Now we will get some data-bound controls into the template.

   In the top left cell, we will place the Northwind logo. Prepare by copying the `NorthwindLogo.bmp` file from the Images folder of the download into the `ch01` folder of your site (`C:\Websites\BegAspNet2Db`). (You can use any other bmp or gif if you don't have the download). Into the top left cell, drag an image control from the Standard section of the toolbar. Select the image control and in the properties window set the ImageUrl to `NorthWindLogo.bmp`.

Into the top right cell, drag a label from the toolbar and click Edit Data Bindings in the smart task panel. Bind its text to the category ID field and click OK.

Into the bottom left cell, drag a label and bind it to the description field.

Into the bottom right cell, add a label and bind to the category name field.

7 Now that the cells are filled, select the right column by clicking on the arrow that appears when you locate the mouse cursor right at the top of the column. Once selected, in the properties window, set align to center. Finish by opening the smart task panel (arrow at top right of entire DataList control) and clicking on End Template Editing. Save the document (which will appear as follows) and press Ctrl+F5 to run it. Your page should resemble the following code:

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<script runat="server"> </script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Chapter 1 TIO #2 DataList From Access</title>
</head>
<body>
    <h2>
Chapter 1 TIO #2 DataList From Access</h2>
    <form id="form1" runat="server">
         <asp:DataList ID="DataList1" Runat="server"
DataSourceID="AccessDataSource1"
            DataKeyField="CategoryID" RepeatColumns="3"
RepeatDirection="Horizontal">
            <ItemTemplate>
                <table><tr>
                        <td style="width: 100px" bgcolor="aqua">
                             <asp:Image ID="Image1" Runat="server" />
                        </td>
                        <td style="width: 100px">
                            <asp:Label ID="Label1" Runat="server"
Text='<%# Eval("CategoryID") %>'></asp:Label>
                        </td>
                    </tr>
                    <tr>
                        <td style="width: 100px" bgcolor="aqua">
                            <asp:Label ID="Label2" Runat="server"
Text='<%# Eval("Description") %>'></asp:Label>
                        </td>
                        <td style="width: 100px">
                            <asp:Label ID="Label3" Runat="server"
Text='<%# Eval("CategoryName") %>'></asp:Label>
                        </td>
                    </tr></table>
                <br />
                <br />
            </ItemTemplate>
        </asp:DataList>
```

```
<asp:AccessDataSource ID="AccessDataSource1" Runat="server"
                DataFile="~/Data/Northwind.mdb"
                 SelectCommand="SELECT * FROM [Categories]">
        </asp:AccessDataSource>
    </form></body></html>
```

How It Works #2 — DataList from Access

You can see the same patterns here as in the first exercise. You do not write any code, and the page is in HTML format. All of your actions are drag-and-drop or selections, so you build a page in just a few minutes. As before, you added a DataBound control (DataList) and it automatically walked you through setting up its data source control (AccessDataSourceControl).

The DataList control starts the same way as the GridView, by specifying its DataSource control, but to build a less rigid rendering you go into the control's item template. On this template you can add and format controls and bind them to fields in the underlying DataSource control. The templated controls will then display the appropriate value for their record in the DataList. In this case, you started with a table to organize the space in the template and filled the cells with three labels. When the DataList is rendered you see a more flexible layout than you had with a GridView. All of the topics discussed in this demonstration will be fodder for deeper analysis in later chapters of the book.

Try It Out #3 — TreeView from XML

Your sources of data are not limited to relational databases, and your data displays do not have to be in rectangular grids. In this exercise you read data from an XML file and display it in hierarchical fashion in a TreeView control. You will also see how to handle an event on a TreeView control when a selection is made, one of the few cases where you have to write code.

1. Copy the file named `Bookstore.XML` from the downloads into your Data folder. Then create a sister folder to Data named Images and copy in three gifs from the download: `closedbook.gif`, `notepad.gif`, and `folder.gif`.

2. Create a new Web Form page in `ch01` named `TIO-3-TreeViewXML.aspx`. In Design view, drag an XMLDataSource control from the Data section of the toolbox onto the page and then click Configure Data Source from the common tasks menu. You only need to fill in two spaces (double check your syntax).

```
Data File:                    ~/Data/Bookstore.xml
XPath expression:             Bookstore/genre[@name='Fiction']/book
```

3. Now you need to display the data, so drag a TreeView control from the Standard section for the toolbox onto the page and set its Data Source to XMLDataSource1 (the default name for the control you created in the last step). Save and run the page at this point to see a tree of the generic names of the XML nodes (book, chapter), but not the actual values (Tale of Two Cities, London).

4. Exit your browser and go back to the page in Design view. Open the CommonTasks menu for the TreeView and click on EditTreeNode DataBindings. In the available list click Book and click Add. Set three properties for book, as follows:

```
DataMember:     Book
TextField:      Title
ImageURL (careful - NOT ImageUrlField):       ~/Images/closedbook.gif
```

5. Staying in the DataBindings DialogBox, add the chapter node and set its DataMember to chapter, TextField to name, and ImageURL to notepad.gif. Apply and click OK to close. Save the page and view it in your browser.

6. You will add one last feature to demonstrate that you can handle events in a similar fashion to older versions of ASP. Exit your browser and go to VWD. In Design view, select the TreeView control. Display the properties window with F4. At the top of the properties window, click on the lightning bolt and then double-click on `SelectedNodeChanged` event. VWD will automatically type the first and last lines of an event handler. You can now type in the highlighted line below. The first option uses Visual Basic.

```
<%@ Page Language="VB" %>
<script runat="server">
  Sub TreeView_Select(ByVal sender As Object, ByVal e As EventArgs)
    Response.Write("You selected: " & TreeView1.SelectedNode.Value)
  End Sub
</script>
```

Option two employees C#, as follows, if you chose to use C# when you created the page.

```
<%@ Page Language="c#" >
<script runat="server">
    void TreeView_Select(Object sender, EventArgs e)
    {
        Response.Write("You selected: " + TreeView1.SelectedNode.Value);
    }
</script>
```

7. Last, you need to instruct the TreeView to use this event handler when any value of the tree is clicked. Staying in Source view, add the following gray line to the <asp:TreeView> control:

```
        <asp:TreeView ID="TreeView1" Runat="server"
               OnSelectedNodeChanged="TreeView_Select"
              DataSourceID="XmlDataSource1">
```

8. Save the file and run it. When you click on any item in the tree the event is triggered and the page displays a small note repeating the value of the text under the click. The finished file looks like the following using Visual Basic:

```
<%@ Page Language="VB" AutoEventWireup="false" ClassName="TIO_3_TreeViewXML_aspx"
%>
<script runat="server">
    Sub TreeView_Select(ByVal sender As Object, ByVal e As EventArgs)
        Response.Write("You selected: " & TreeView1.SelectedNode.Value)
    End Sub
</script>

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

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Chapter 1 TIO #3 TreeView from XML</title>
</head>
<body>
```

```
    <h2>
Chapter 1 TIO #3 TreeView from XML
    </h2>
    <form id="form1" runat="server">
    <div>
        <asp:XmlDataSource ID="XmlDataSource1" Runat="server"
            DataFile="~/Data/Bookstore.xml"
            XPath="Bookstore/genre[@name='Fiction']/book" />

        <asp:TreeView ID="TreeView1" Runat="server"
            DataSourceID="XmlDataSource1"
            OnSelectedNodeChanged="TreeView_Select" >
            <DataBindings>
                <asp:TreeNodeBinding ImageUrl="~/Images/closedbook.gif"
TextField="Title" DataMember="book" />
                <asp:TreeNodeBinding ImageUrl="~/Images/notepad.gif"
TextField="name" DataMember="chapter" />
            </DataBindings>
        </asp:TreeView>

    </div>
    </form>
</body>
</html>
```

The script part of the finished file looks like the following using C#:

```
<%@ Page Language="c#" AutoEventWireup="false" ClassName="TIO_3_TreeViewXML_aspx"
%>

<script runat="server">
    void TreeView_Select(Object sender, EventArgs e)
    {
        Response.Write("You selected: " + TreeView1.SelectedNode.Value);
    }
</script>
...
```

How It Works #3 — TreeView from XML

The same themes emerge in this third example. You create a data source control (XMLDataSource) and a data-bound control (TreeView). By default, the tree shows the names of the nodes rather than their values. But even at this level, you have achieved an expanding and collapsing tree with just a few clicks. Then, in the Edit TreeNode Bindings dialog box, you determined what values and images to display at each level.

Note two items that you will cover in more depth later. In specifying the XML file, you started the path with a tilde (~). This represents the root of the Web site. If the site is deployed to another physical location on a drive, the link will not be broken. Second, note how the XPath string is used to limit the number of books displayed. There are five titles in the XML file. If you change the central part of the XPath to `[@name='NonFiction']`, you can see the other two books. Last, note that XPath is case-sensitive.

The last exercise ended with a little coding. As in earlier versions of ASP, you can set an event to fire an event handler with custom code. In this case, the TreeView has an `OnSelectedNode` event that will send the value of the node that was selected to the event handler. You can display that with a simple Response.Write.

Summary

This chapter reviewed some basic ASP.NET topics, set up your machine for the book, and walked through three exercises.

Recall from your study of ASP.NET that pages are stored as a combination of text, HTML tags, scripts, and ASP.NET server-side controls. When a page is requested, IIS directs the request to ASP.NET, which processes the page using the ASP.NET controls and scripts to build a pure HTML page. Therefore, you achieve two objectives. First, you can build each page to suit the individual requestor and the current needs of the business (the pages are dynamic). Second, any browser can read the page because it is delivered as pure HMTL.

The .NET Framework is a collection of classes (code) written by Microsoft that enable programmers to quickly, surely, and easily create solutions for their organization. One set of classes, ASP.NET (System.Web), provides very powerful controls for building dynamic Web pages. ASP.NET 2.0 reduces the amount of coding for common data scenarios to near zero. All of the decisions about ADO object parameters and the timing of bindings are handled automatically. The page architecture follows a pattern of having two controls to show data. The first is a data source control that establishes a connection to a database. The second is a data-bound control that displays the data.

This chapter discussed various software options to implement data on ASP.NET 2.0 pages. The .NET Framework 2.0 must be installed. You have two Web server options. Although public deployment would use IIS, development explained in this book uses the lighter-weight Cassini that comes with Visual Web Developer. To create pages you will be using Visual Web Developer Express because it offers many tools in its Integrated Development Environment (IDE) and is available for free. Several alternatives for a database management system were discussed. This book uses SSE because it is free, overcomes the problems of using Access, and easily scales to Microsoft SQL Server at the time of deployment. Last, you installed two databases (Northwind and Pubs) that you will use for examples in this book.

The exercises in this chapter revealed that it takes just a few ASP.NET tags with a half dozen attributes each to display and modify data. You used data source controls (SqlDataSource, AccessDataSource, and XMLDataSource) to create a conduit to the data. Then you used data-bound controls (GridView, DataList, and TreeView) to display the data. You did not have to use any <script> code to perform the data-binding tasks, although you did demonstrate that you could write an event handler as in earlier versions.

One control can have its parameters set by another control, for example, when the data source for the GridView was changed by the selection in the ListBox. Those parameters were passed easily across pages as when you sent a record ID to the details page. Overall, understand that none of these pages would take more than ten minutes to type and troubleshoot using the drag-and-drop tools of Visual Web Developer.

The next few chapters will examine data source controls, starting with Access, moving on to the SQL data source control, and then connecting to other databases. Following that, several chapters will discuss displaying data using various data-bound controls.

Exercises

1. What is the basic pattern for showing data on an ASP.NET 2.0 page?
2. Name several types of data sources that ASP.NET 2.0 supports.
3. Name several ways that ASP.NET 2.0 can display data.
4. What is the difference between the .NET Framework 2.0 and ASP.NET 2.0?
5. Make some observations comparing SQL Server, MSDE, and SSE.