1

Introduction to ASP.NET 2.0 and ADO.NET

The speed and power that ASP.NET 2.0 and the .NET Framework 2.0 bring to the beginning programmer make it a joy to write this edition. We believe you will find many moments when you just won't believe the quality of results you can achieve with so little effort. We think you will often say, as other programmers did at the first demos, "Wow, this is the way it should be." 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 in an ASP.NET 2.0 Web page, an explanation of 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 reviews knowledge covered in the prerequisites for a basic knowledge of ASP.NET and familiarity with basic database tasks). The sections are organized as follows:

- Overview of the technologies, including the .NET 2.0 Framework, ASP.NET 2.0, and ADO.NET; a short review of how the older versions (1.*x*) worked with data; and a review of some key terminology
- Discussion of the components needed to use data on ASP.NET 2.0 pages
- Setup for this book
- Demonstration of several ASP.NET 2.0 pages that utilize data

As in previous editions, we include a list of common mistakes, a thorough summary, and some questions for you to check your knowledge.

Overview of the .NET 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, a version that promised to decrease the number of lines of code

required to create ASP.NET pages by 70 percent. Such a significant increase in productivity 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 exceeded 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 in 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, 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 capability to serve up their information in XML, making them appealing to all .NET data consumers. .NET supports other standards for information exchange (e.g. t-SQL, cryptographically strong streams, and non–XML-compliant HTML). But in the beginning, you will most frequently need to connect to XML.

.NET supports the Web Services standard for software to request the running of code in remote software using the open-platform standard Simple Object Access Protocol (SOAP) and the language 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, a set of programming tools, and a set of application program interfaces (APIs), called the .NET Framework, to enable the development community to build client-server applications, .NET client and server 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 capability 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.NET, or Cobol while leveraging all features in the .NET Framework.

The .NET Framework also includes a set of *class libraries*, which are large sets of code that can be used by programmers. They provide common functionality that applications may need. 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:

- **System Types *(System)*:** Includes definitions for most simple data types, conversion functions between types, some mathematical functions, arrays, and ways to handle exceptions.
- **Input/Output *(System.IO)*:** Includes classes to read and write to data streams (for example, output to an external device such as a Pocket PC) and to files on drives.
- **Data Access *(System.Data)*:** Includes classes to interact with external data, typically a database. This namespace includes most of ADO.NET, which is used by the ASP.NET 2.0 data controls, and thus is of particular interest to this book.
- **Security *(System.Security)*:** Includes the classes to implement the foundations of security, particularly the system of permissions.
- **Data Structures *(System.Collections)*:** Includes classes to organize and maintain sets of data within .NET including lists, dictionaries, and hash tables. (Note that the System.Data namespace is for communicating with sets of external data.)
- **Configuration *(System.Configuration)*:** Includes classes to establish configuration settings and to handle errors using the settings.
- **Networking *(System.Net)*:** Includes classes to work with networking protocols such as IP and Sockets.
- **Reflection *(System.Reflection)*:** Includes classes that allow a program to look at itself to identify its own characteristics, such as specialty data types and the organization of the code.
- **Globalization *(System.Globalization)*:** Includes classes that hold culture-specific settings such as syntax of dates and currency.
- **Painting and Drawing *(System.Drawing)*:** Includes classes that can interface with the graphical user interface. This namespace includes many classes for specific types of renderings, including Pen, Brush, and Imaging.
- **Tracing and Diagnostics *(System.Diagnostics)*:** Includes classes to analyze problems, including the dissection of system processes, process counters, and event logs.
- **Windows (Client) Application Model *(System.Windows.Forms)*:** Includes classes that allow implementation of the Windows user interface.
- **Web Application Model *(System.Web)*:** Includes classes that provide a programming model for producing Web pages. The Web page code runs on the server and renders HTML to the client's browser. This class includes over a dozen subclasses that focus on various types of browsers and server services. These are of great interest in this text because System.Web is, essentially, ASP.NET.

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. Looking at the above list, we can see that 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. ASP.NET 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 files, which are served directly from the Web server, ASP.NET pages are built on the server to produce dynamic results. The final rendering of a page might be constructed from a variety of different instructions or data sources.

ASP.NET pages are created by a programmer as a combination of text, markup (such as HTML), and ASP.NET server-specific tags and script, which are then stored with the `.aspx` extension 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 HTML that the client browser can display. Because the rendered output is HTML markup (no proprietary code), any browser can read it. All the dynamic processing happens on the Web server. ASP.NET server-specific tags are very powerful; they include the capability to react to user actions, connect to data stores, and automatically build very complex HTML structures.

As previously mentioned, ASP.NET represents one 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 rich Web applications more easily than ever before. You can spend less time reinventing the basic building blocks that all applications need and instead spend more time focusing on the specific logic that is unique to your application.

ASP.NET also introduces Web programming innovations 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 Visual C#, Visual Basic .NET, or J#. Classic ASP (versions 1, 2, and 3), on the other hand, was generally used only with 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. Frequently, an ASP.NET page will run an order of magnitude faster than the same page written in Classic ASP.
- **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 experience in Classic 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 to 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 constructed a table of data from a database query?) One of the great advancements that ASP.NET brings to Web programming is Microsoft's encapsulation of code to perform common 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 over 80 server controls that encapsulate behavior that ranges 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 the Smart Tasks panels, tag-level navigation bars, and wizards that can set control properties.

If you have worked in Classic ASP, you will be amazed at the increase in your productivity with ASP.NET 2.0. If you have never worked in Classic ASP, you can buy a beer for someone who did and listen to the war stories.

Introduction to ASP.NET 2.0

The first ASP.NET versions (1.0 and 1.1) rapidly spread throughout the developer community from a pre-release in 2001 to 2005. 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 Web developers.

However, even prior to the release of version 1.0, the ASP.NET team was already working on ASP.NET 2.0. The team set 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 interfaces that expose those services with a minimum of code.
- Improve page-processing performance to increase the overall Web server throughput.
- Provide administration features that enhance the deployment, management, and operations of ASP.NET.
- 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.

Let's pause to reflect on that first goal: the removal of 70 percent of the code needed today to write a dynamic Web application. How is this possible? The ASP.NET team 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 (server controls or classes that provide services) that could accomplish those tasks automatically. For example, most Web applications need security, 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. By providing these class libraries, Microsoft greatly reduced the amount of code that must be written by programmers to implement common scenarios.

Among all the common scenarios, however, one stood apart as absolutely essential to every application. Data access is the common thread that drives most 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. The goals that were specific to data include the following:

- 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 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 capability to both display and manipulate data.
- Enable an extensible model for building custom data sources to support new types of data.
- Enable an extensible model for building custom controls for displaying data in new ways.

The result of the preceding objectives is a set of server controls that programmers can use 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 connection to stores of data (databases or other repositories). 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 combinations.

A number of data source controls are available for many types of databases and even nonrelational data sources. Although additional controls are already in development by third parties, the following five data source controls ship with ASP.NET 2.0:

- **SqlDataSource control:** To connect to the Microsoft SQL Server and other T-SQL databases that have a managed provider
- **AccessDataSource control:** To connect to Microsoft Access files (extension of `.MDB`)
- **ObjectDataSource:** To connect to middle-tier business objects
- **XmlDataSource:** To connect XML content
- **SiteMapDataSource:** To connect XML files in the format of the ASP.NET 2.0 site map (or site data through a custom provider)

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:

- **GridView and DataGrid:** For tabular data. `GridView` is new for version 2, whereas the legacy `DataGrid` remains largely unchanged for ASP.NET 1.*x*.
- **DetailsView and FormView:** Display data for one record at a time.

- **DataList and Repeater:** Display multiple data records in flexible layouts defined by the page programmer.
- **ListBox, DropDownList, BulletedList, CheckBoxList, and RadioButtonList:** Display a group of options with the expectation that the user will make a selection.
- **AdRotator:** Display an advertisement from a store of advertisements. This control was offered in ASP.NET version 1.*x* but has been updated to be compatible with data source controls.
- **TreeView and MenuView:** Display hierarchical data.

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

Introduction to ADO.NET

Working behind the scenes in all versions of ASP is ADO, the ActiveX Data Objects. *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, including ASP. In this section we will take a short look at ADO.NET so you have some feel for the technology. Keep in mind that for most scenarios in ASP.NET 2.0, you do not need to work directly with ADO.NET.

The classes within ADO.NET handle 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. ADO.NET, like ASP.NET, is not a language; rather, it is a collection of classes that provide common functionality. You can use those classes from a programming language such as Visual Basic or C#.

You can think of ADO.NET as a very smart translation layer between a data store (like SQL Server) and a data consumer (like an ASP.NET page). ADO.NET classes can generate commands that are appropriate to carry out tasks in the data store. 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.

Review of ASP.NET 1.x and ADO.NET for Data Access

This section is one of the few places where we discuss ASP.NET 1.*x*, meaning version 1.0 or 1.1. We do so for purposes of comparison. Pay close attention in each sentence to its application for ASP.NET 1.*x* (last version) versus ASP.NET 2.0 (the topic of this book).

Many readers will already have experience with earlier versions of ASP.NET. This short section recalls that model for the purposes of demonstrating how, in version 1.*x*, you worked directly with the ADO.NET objects to bring data into a Web page. For those readers who never used earlier versions, view the code that follows 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. Note that although the following is presented only in Visual Basic, the rest of the techniques in this book are covered in VB.NET and C#.

```
<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_lname] 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 `SQL SELECT` statement that reads the information about some authors from a database and then 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 capability 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 create itself from 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

Having endured a long winter of discontent with ASP.NET 1.*x*, we can now turn to the pearl of ASP.NET 2.0. Version 2.0 gives us an improved set of tools for data access that 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.NET objects as in the preceding listing. Instead, you can add simple server-side data controls to your page and set their attributes declaratively. 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 previous 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='<%$ ConnectionStrings: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>
```

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.NET 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.

The second improvement occurs because the server-side controls are sensitive to events in the page life cycle. Appropriate actions are taken by the ASP.NET 2.0 server-side controls during the page life cycle. 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.NET were typically confused by the intricacies of when in a page life to perform various tasks, such as data-binding. Thus, many ASP.NET 1.*x* pages suffered from code that either called `DataBind` in 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.

Review of Terminology

To round out the introductory material, this section provides a review of some of the terminology used throughout the book.

- **Access:** An RDMS that is based on the MDB (Access) 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.
- **ADO.NET:** A collection of classes that acts as an intermediary between data stores (such as Access or an XML file) and a data consumer (such as an ASP.NET page).
- **ASP.NET:** A set of class libraries in the .NET Framework for building and running dynamic Web applications.
- **Command:** An ADO.NET object that represents a SQL statement that can be passed to a database.
- **Common Language Runtime (CLR):** The .NET runtime environment. It enables programmers to write in one of many languages, and then compile the code to a single, uniform language for deployment.
- **Connection:** An ADO.NET object that represents a unique path between a data consumer and data provider.
- **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.
- **Data store:** A place where data is kept and usually managed. All RDMS are data stores, but some data stores are not RDMS 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 RDMS must store data in compliance with some form of normalization (a relational format). These databases include SQL Server, Oracle, Access, and the other main database products.
- **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.
- **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.
- **Data source control:** A server-side control that creates a connection to a database. It provides an abstraction of the data store and makes programming ASP.NET 2.0 pages faster and easier to build. Data source controls are available for Microsoft SQL Server, Microsoft Access, XML, and other sources of data.
- **DataSet:** An ADO.NET object representing a group of data organized into rows and columns.
- **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.

- **Extensible Markup Language (XML):** A markup language commonly used 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.
- **Integrated Development Environment (IDE):** A set of tools that assists programmers in developing code.
- **Internet Information Server (IIS):** A built-in Web server in Windows that serves Web pages to requestors via TCP/IP. IIS operating on Windows 2000, 2003, or Windows XP Professional has the capability to use the .NET 2.0 Framework classes to serve ASP.NET 2.0 Web pages.
- **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.
- **.NET Framework:** A group of classes written by Microsoft that make applications easier and faster to develop and more suitable for operation over the Internet.
- **Parameter:** An ADO.NET object that represents a variable piece of data that can be sent to the SQL Server with the `Command Object` (SQL statement) and inserted by the SQL Server prior to execution of the statement
- **Server control:** An object that derives from `System.Web.UI.Control` that performs its tasks on the server to contribute to an HTML page that is sent to the browser. Server-side controls can maintain a state through `ViewState`.
- **SQL Server:** An enterprise-strength RDMS designed to support large amounts of data and many concurrent users.
- **SQL Server Express (SSE):** A database engine (currently free) 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.
- **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 an RDMS.
- **ViewState control:** A string (frequently very long) that describes the current state of the page and is posted back to the server when the page is refreshed. The server uses the `ViewState` string to then take the next appropriate action for the user.
- **Visual Studio (VS):** A very powerful IDE that is optimized for .NET development. Visual Studio includes an editor to design pages, an Explorer-like file manager, and many tools increase developer productivity.
- **Visual Web Developer Express (VWD):** Another IDE from Microsoft that is similar in look and feel to VS but with a reduced set of features. Currently VWD Express is free, so we use it in this book.
- **Web Page Editor:** Software that allows pages to be opened and changed. One of the most basic editors is Notepad. Visual Studio, Visual Web Developer, and ASP.NET Web Matrix include an editor with other tools to improve productivity.

With these past few sections, we have defined ASP.NET 2.0 and its importance. Now, we will describe the prerequisites and then we can move on to creating our first few pages.

Requirements for Using ASP.NET 2.0

This section describes and discusses the options for the components you need to add data to your ASP.NET 2.0 pages. The next section steps you through the actual installations of the four prerequisites needed for using databases with ASP.NET 2.0:

- A Web server
- The .NET Framework version 2.0
- An editor to create ASPX pages
- Database management system

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. IIS handles the load of a public Web site, but may also be used for development purposes. If you have already turned on IIS on your development machine, you will find that the .NET Framework automatically registers itself with IIS and is available for you to create ASP.NET pages.

If IIS is not turned on, you can easily enable it in Windows:

1. Check that you have updated your installation of Windows with the latest service packs and security patches. (This is a good habit to get into.)
2. Click through Start➪Control Panel➪Add & Remove Programs.
3. Select Add/Remove Windows Components, add a check mark to IIS, and click OK to install.

Microsoft also provides a lightweight alternative to IIS named the ASP.NET Development Server (code-named Cassini) that is better suited to developers and students than IIS. The ASP.NET Development Server comes with Visual Studio and Visual Web Developer and installs automatically. When a page is run from the Visual Web Developer, ASP.NET Development Server 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`. The `xxxx` represents a random number of the port that is opened for the Web server (the number is randomized to prevent people from searching the Internet for developers who happen to have a specific port open at that moment). Instead of IIS, ASP.NET Development Server will serve the page, including processing ASP.NET code and server controls. Evidence of the server activity appears as an icon system tray.

> **ASP.NET Development Server is designed for developers and does not support requests from remote computers. 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.**

The .NET Framework Version 2.0

The .NET 2.0 Framework includes the classes that enable ASP.NET 2.0, including its data connection capabilities. The framework is available for download in three ways:

- The first is the version named .NET Framework 2.0 Redist, which is about 20MB.
- The second version includes the Software Development Kit (SDK), which 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 2.0 Framework and order the SDK on CD, 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 easiest and is adequate for this book. A description of the install of VWD Express follows.

If you have the .NET Framework version 1.0 or 1.1 installed, you can also install 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 is already registered in IIS, the .NET Framework 2.0 setup will not *automatically* register 2.0 with IIS because this would potentially upgrade applications to the new version without the user's consent.

In order to register .NET 2.0 Framework if version 1.*x* is installed, run in a command prompt window `aspnet_regiis -i` from the .NET 2.0 Framework installation directory; for example, `\WINDOWS\Microsoft .NET\Framework\<version>`. This command line utility will upgrade the server and all apps on it to use 2.0.

> **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.**

An Editor to Create Web Pages

ASP.NET pages must be written using some type of editor. Most readers will select one of the following three options, depending on their budget and interest in learning to use new software. This book uses Microsoft 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 10 or 20 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 types all of the tags, attributes, and code to produce the feature. VS also provides intelligent assistance to typing so that you generally have to type only a character or two and VS will complete the syntax (called "IntelliSense" technology). Debugging features, including trace and immediate windows (called stacks), are built into the product. For readers of this book, a key VS feature is the capability 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 for multiple programming languages and tools for collaboration of multiple developers at a price starting 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 and 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 ASP.NET Development Server, the visual design interface, all of the controls we 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 pages. Remember that ASPX files are saved as text files, as are HTML files. Therefore, every exercise in this book can be typed into a text file using Notepad, saved with an *.aspx* extension (not `.txt`), and run. 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.

Additional editor options are likely to appear. Microsoft may add ASP.NET 2.0 capabilities to FrontPage, its Web designer–oriented site development tool. Also, editors may be forthcoming from third parties. However, as long as VWD Express is a free download, its advantages far outstrip the other options. This book uses VWD Express for all exercises and demonstration figures. The resulting code is then generally presented so you can see the source and so typists can enter the page into Notepad or another editor.

A Database Management System

A Web site needs a way to manage the data. ASP.NET 2.0 files will not, by themselves, hold or manage data. Several choices are listed here; we use SQL Server Express (SSE) in this book.

- **Microsoft Access:** Familiar and already widely deployed but not recommended for use in a production Web site. Access works for learning and development purposes, however (more details follow).
- **Microsoft SQL Server:** A powerful choice for public deployment but expensive and more difficult to set up and manage, particularly on a desktop development machine.
- **SQL Server Express (SSE):** A lightweight and free database engine based on the SQL Server engine. SSE is used to support this book's examples in most cases.
- **Other relational databases:** Examples that can be used include Oracle or MySql. This book discusses using their data but not how to install or manage them.
- **Nonrelational stores of data:** Examples include text files, XML files, or Excel spreadsheets, in some cases.

Let's look at some details for each of these five options.

Microsoft Access

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

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

When using Access as a source of data for Web sites, even in development environments, you should be aware of two major problems:

- Access was not designed to scale to large numbers of users. 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 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, we 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.

SQL Server Express

The SQL Server Express (SSE) edition provides an engine that holds a database and responds to T-SQL statement commands. SSE is built from Microsoft SQL Server and, thus, responds exactly like the full-scale SQL Server. SSE is the next generation to Microsoft Data Engine (MSDE), and the upgrade path to SQL Server is seamless. As of this writing, SSE 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. Keep your eyes open for Microsoft to add a basic GUI interface. If your machine has SQL Enterprise Tools on it (part of the SQL Server software), then you can use it to manage SSE instances. There are several advantages to SSE over other sources of data for readers of this book:

- SSE is free and installs automatically with VWD.
- Developers spend 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 importing SSE databases to SQL Server.

The biggest disadvantage is that as of this writing, there is no GUI interface for installing or importing a new database. We will talk you through the steps of installing your sample databases for this book using VWD Express.

Other Relational Databases

ASP.NET 2.0 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.mbd` 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.

Nonrelational Stores of Data

Another option is data stored in files or streams. For example, XML has emerged as an important format for storing and transferring data. Other data formats, such as Excel, can also be used in special cases. These file-based data stores exist independently of any database management system. They can be used directly by ASP.NET 2.0. In the rare cases that these are the only sources of data, there is no need to install a database management system.

Now that we've covered the components needed at a theoretical level, it is time to get out the keyboard and actually set up your machine for the exercises in this book.

Setup for This Book

We've discussed the software needed for running, creating, and testing data-driven ASPX pages, and it is now time to walk you through the setup. We will also perform some steps that are specific for this book, not for ASP.NET 2.0 in general.

Install Visual Web Developer Express, SSE, and the ASP.NET Development Server

Visual Web Developer is a one-stop package for most of what is needed in this book. Its install wizard installs all of the Microsoft-provided components discussed in the previous section: the .NET Framework 2.0, the ASPX development environment including the page editor (VWD Express), the ASP.NET Development Server, and SQL Server Express.

1. Shut down any open 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 .NET Framework. The help library is recommended; the SDK is optional (it provides sample code and FAQs).
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 prompts you to specify the authentication, then select Windows Authentication.
6. Finish the install and reboot.

> **After finishing the setup of this chapter, you may also want to download the set of Quick Start sample applications. They have simple install instructions and give you a useful, applied reference.**

After this step, you can perform two confirmations. Your Start➪Programs will now contain Visual Web Developer 2005 Express Edition. Your Start➪Control Panel➪Administrative Tools➪Services will display SQL Server running. Note that your SSE instance will be named `(local)\SQLExpress`. Another syntax for the name is `.\SQLExpress` (note the leading period). This instance name will come up many times in this book and can be a source of error if not typed correctly. You can now use SSE as your database management system, but keep in mind that the engine does not yet hold a database.

Download This Book's Files

The materials needed for the exercises in this book are available from the publisher:

1. Create a folder to hold the downloads for this book. We suggest you use `C:\TempBegAspNet2DB`. Note that although we park the download files here, the site will be created in the normal folder for IIS's Web applications, `C:\Websites`.
2. Go to `www.wrox.com`, get the download files for this book, and store them in `C:\TempBegAspNet2DB`. Extract the files to the same folder.

The download will contain notes on the install, the sample database setup files, demonstration pages used in the text, and all of the exercises. Read the install notes.

Create the Practice Web Site

Before doing any other steps, we will create the Web site used in this book. The site will be stored in the `C:\Websites` folder (created automatically by IIS) within which we will create a `BegAspNet2DB` folder.

1. Start VWD. You will see a Start Page in the center of the screen that is similar to Figure 1-1 (but with an empty list of recent projects). This is useful in the future, but close it for now.
2. Select File➪NewWebSite to see the New Web Site dialog box, as shown in Figure 1-2. Select the template type ASP.NET Web Site and locate the site in `C:\Websites\BegAspNet2Db`. Accept the defaults for File System and language of Visual Basic.

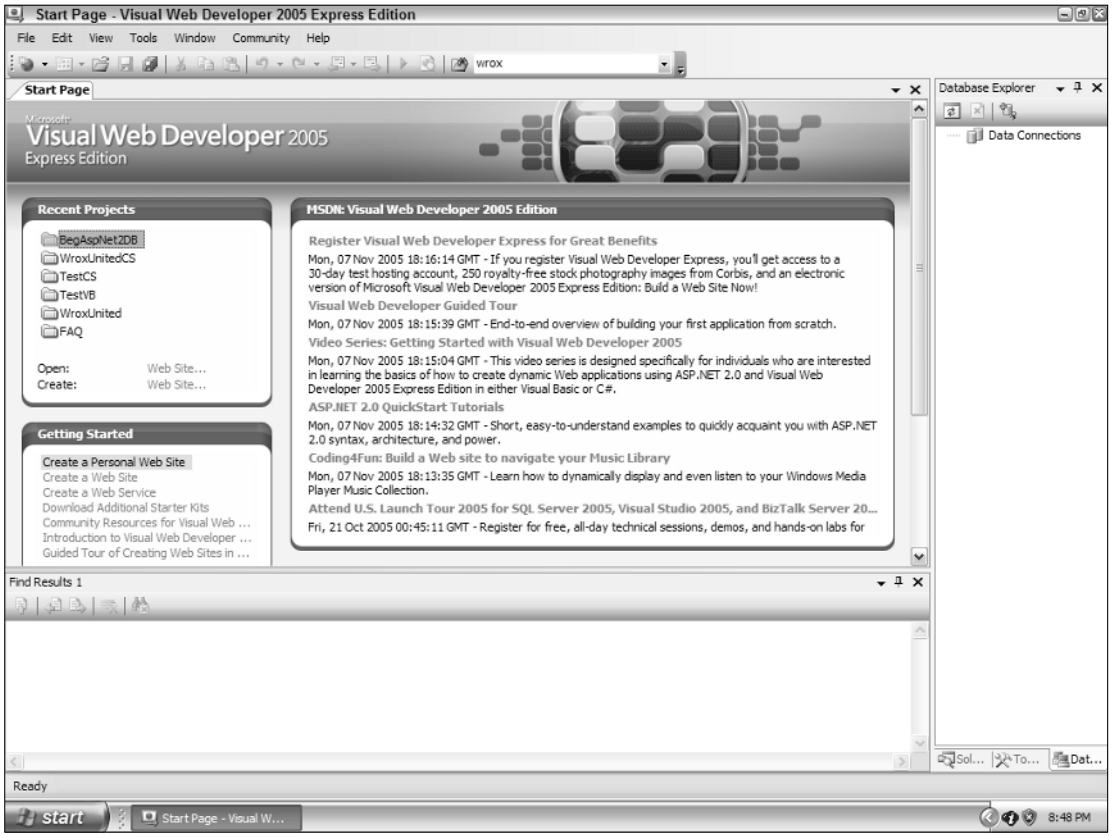

Figure 1-1

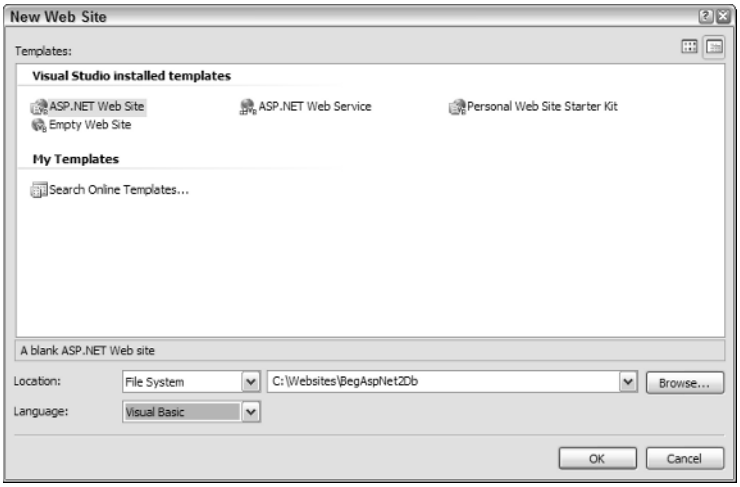

Figure 1-2

3. VWD will automatically build the framework of a site and present to you a default page as in Figure 1-2. In the lower-left corner, click the Design view (WYSIWYG — What You See Is What You Get) that is currently empty or a Source view of the same page (see tabs in Figure 1-3).

Figure 1-3

4. Take a look in the Solution Explorer by selecting View➪Solution Explorer or press Ctr+Alt+L. Note that VWD has created the root and `App_Data` folders. Within the root are the `web.config` and `Default.aspx` pages and a folder for `App_Data` (see Figure 1-4).

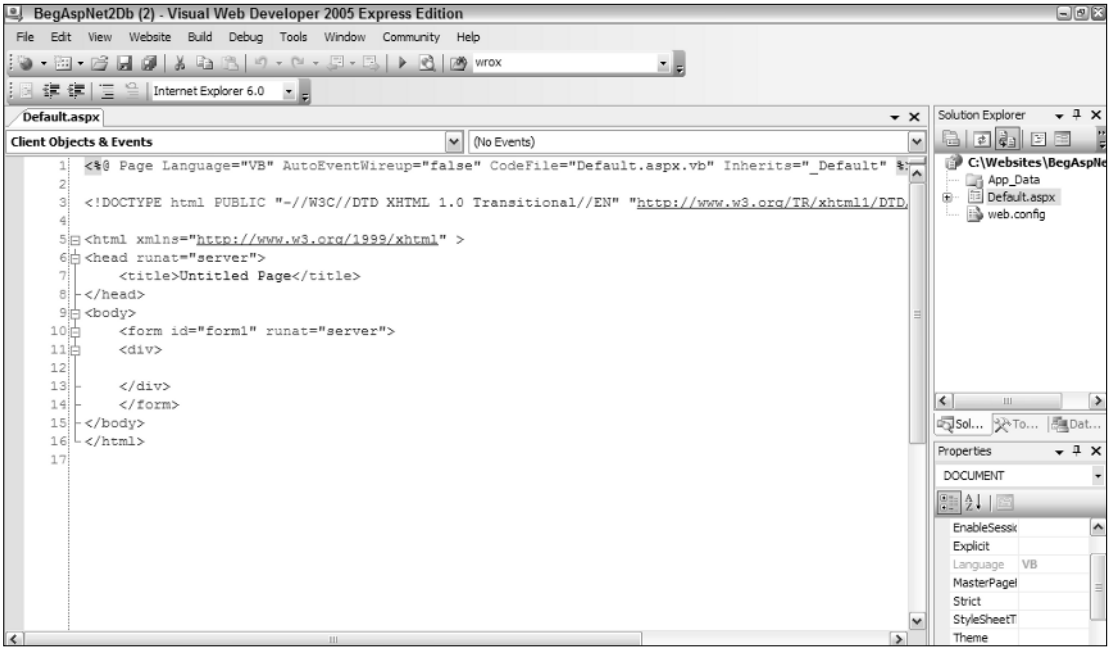

Figure 1-4

Later in the chapter, we will begin adding pages into folders we create named `ch01`, `ch02`, and so on.

Install the Sample Databases

In the download files, we provide two files of data used for data transfer — `Northwind.mdf` and `Pubs.mdf` — the same databases we have all come to know and love through many versions of Access and SQL Server. We also include `Northwind` in Access format for some demonstrations (`.mdb` file). These download files can be added to your site (and explored) in the following steps. First we physically copy the files into our site. Then we establish logical connections to them as sources of data.

1. In VWD's Solution Explorer, right-click on `App_Data` and select add Existing Item. Browse to your download for this book, select `App_Data\Northwind.mdf`, and click OK. Repeat for `Pubs.mdf` and `Northwind.mdb`.
2. Switch to the Database Explorer by clicking the tab that sits, by default, on the right side of the screen, as shown in Figure 1-5. If the tab is not visible, click through the menu: View➪Database Explorer.

Figure 1-5

3. Click on the Connect to Database icon and select Add Connection. In the Add Connection dialog box, select a Data source of Microsoft SQL Server Database file. This is a little confusing because the term "data source" refers to the type of source — that is, SQL format, Access format, and so on. In the Database file name box, browse to `App_Data\Northwind.mdf` and click Open. Accept the default logon using Windows Authentication, click Test Connection, and then click OK.
4. Repeat for the `Pubs.mdf` file both the Add Existing Item in the Solution Explorer and the Add new connection in the Database Explorer.
5. Finally, repeat for the `Northwind.mdb` (Access format) file. In Solution Explorer select Add Existing Item the same as we did for the `.mdf` files. However, in Database Explorer, in the Add Connection dialog box use as a data source "Microsoft Access Database File."

Note that your Data Explorer (Server Explorer) now has connection entries for `Northwind` (two connections) and `Pubs`, which will remain available when you open this site in the future. 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 in `.mdf` (or `.mdb`) format is copied into your `App_Data` directory, you can use VWD as a front end to examine and change the data of a database.

That finishes the setup steps. We will finish the chapter with a few quick demonstrations of the ease and power of using ASP.NET 2.0 to access data on Web site pages.

Demonstrations

To finish off the first chapter, we will create several pages that demonstrate using ASP.NET 2.0 pages with data. We 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 for these exercises. To create these pages, you must have completed all of the install steps earlier in this chapter.

Try It Out — Creating a GridView Table from SQL with Paging and Sorting

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

1. Continue using VWD with your BegAspNet2DB site open. If a Default page is open, you can close it.

2. In the Solution Explorer (if not visible on the right, press Ctrl+Alt+L to turn on), right-click on the name of the site, and click New Folder. Name it `ch01`.

3. Right-click on the new `ch01` folder, and add a new item using the template Web Form. Name it `TIO-0101-GridViewSQL`. Set the language to Visual Basic and leave the other settings at their defaults (Select master page = off and Place code in separate file = off), as shown in Figure 1-6. Once the new page appears on the VWD screen you will see, at the bottom left, tabs for Design and Source views. Switch to Design view.

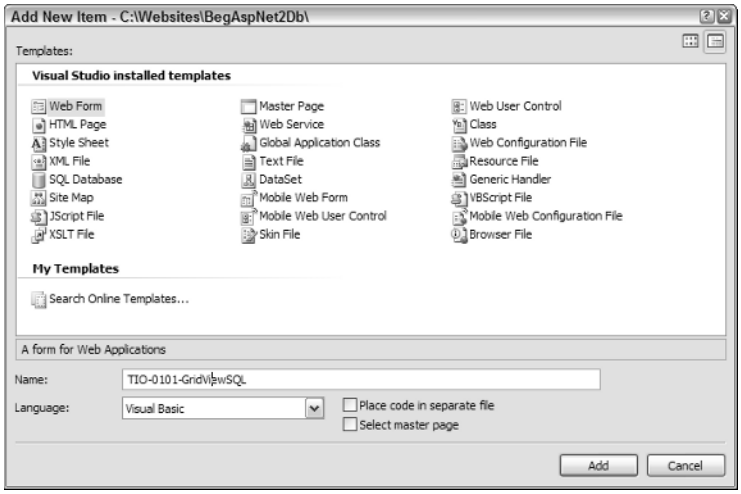

Figure 1-6

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

5. Expand the Pubs connection, Tables folder, and the Authors table as shown in Figure 1-7. Using Control+click, select the four fields `au_id`, `au_lname`, `au_fname`, and `state`. Drag them onto the blank page. VWD will automatically create a `GridView` and a `SqlData Source` control.

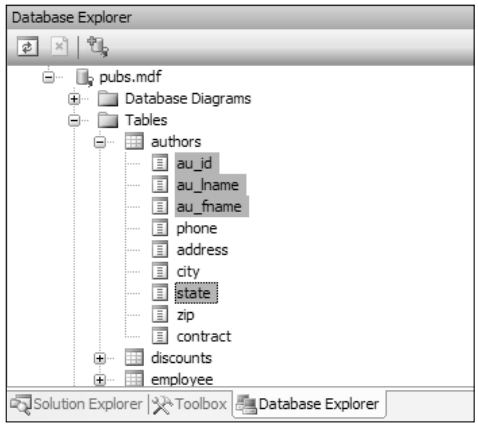

Figure 1-7

6. The Tasks menu automatically opens next to the new control, as in Figure 1-8. (Alternatively, open it by selecting the `GridView` and clicking on the small arrow at the top right of the control.) Enable paging and sorting, and then click AutoFormat and select one that you like (we use Professional). Click Apply and click OK.

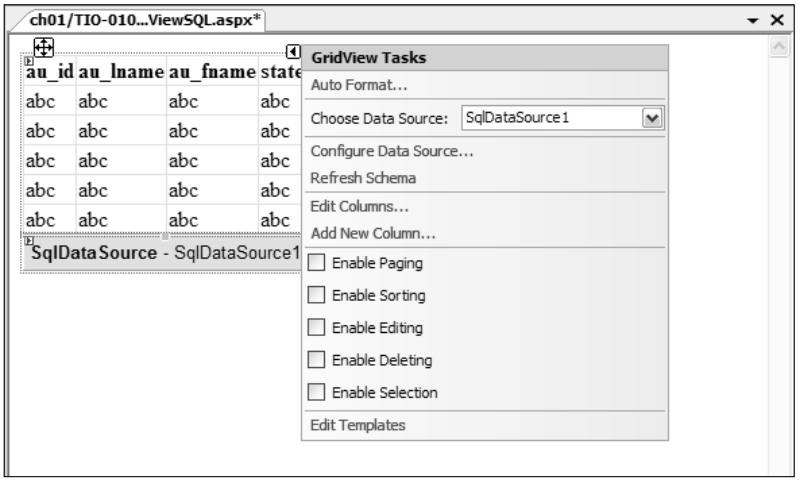

Figure 1-8

7. Save the page with Ctrl+S and press F5 to run it. When warned about debugging being disabled, select to modify the `web.config` file to enable debugging and click OK. In ASP.NET 2.0, the first viewing of a page is delayed as the framework compiles and optimizes the code. Subsequent requests are much faster.

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

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

9. An alternative way to show data uses two steps. First, create the data source control. Then create the data-bound control. Although slower than the drag-and-drop, it provides more control over the process. In prelude, we will create a little working room above the `GridView` by pressing Ctrl+Home, pressing Enter twice, and then pressing Ctrl+Home again. In the next steps, we will first add a data source control and then add the data-bound control

10. 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, as shown in Figure 1-9.

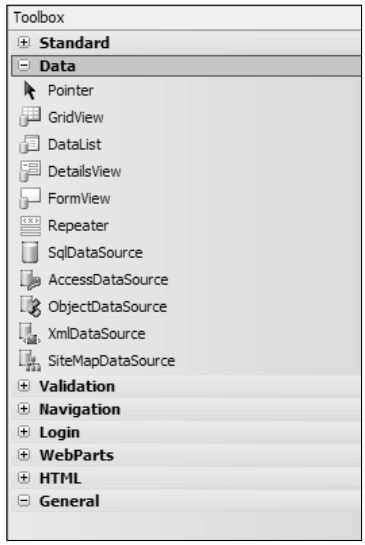

Figure 1-9

11. The Smart Tasks panel will automatically open, as shown in Figure 1-10; click Configure Data Source. Drop down the list of connections, select `pubsConnectionString1`, and click Next. Select the table named `Authors`, and click 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.

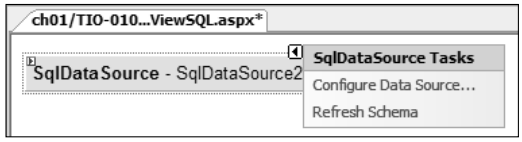

Figure 1-10

12. 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 Tasks panel, click Choose Data Source and select `SqlDataSource2` (the distinct states list). The display and value fields can both remain "state," as shown in Figure 1-11. Click OK. Back on the Smart Tasks panel, click 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.

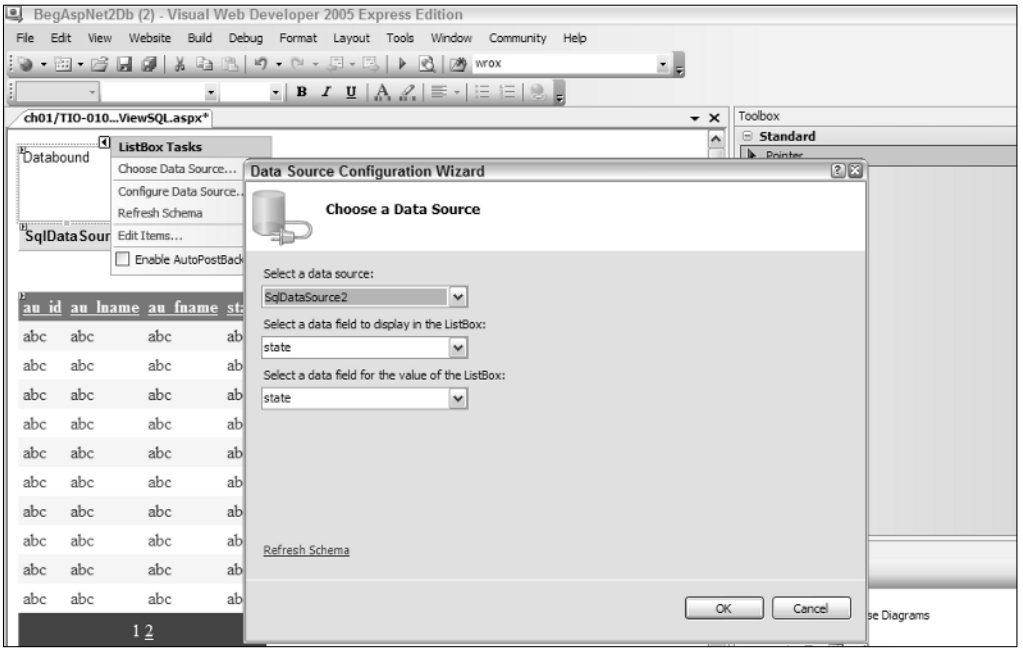

Figure 1-11

13. This step links the list box to the `GridView`, so only authors from the selected state appear. Stay in Design view, and select the `SqlDataSource1` control (below the `GridView` — be careful you don't use the `SqlDataSource2` control) and open its Smart Tasks panel. Click Configure Data Source, keep the same connection (named `pubsConnectionString1`), and click Next. You should see the option button set to specify columns and check-offs for the four fields (`id`, `firstname`, `lastname`, and `state`).

    Stay in the wizard, and click 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. Click OK and click Next.

    In the next page of the wizard, test the query using the default value of CA. Click Finish and then close the Common Tasks menu. If you are asked to refresh the control, click Yes.

14. Save and then run the file. Select various states and note the change in the table. Close the browser when you are done.

15. Now, you will create a second page to display details about one author from our `GridView`. In VWD, click your Solution Explorer tab so it is on top (or press Ctrl+Alt+L). Right-click on `ch01` and add a new item of template type Web Form and name it `TIO0101-DetailsViewSql.aspx`. Accept the other defaults and click `Add`. At the bottom left of the screen, select the Design view. Drag and drop a `DetailsView` control from the toolbar (Data section).

The Smart Tasks panel opens. Choose a New data source and select the type of database (which, by default, means a Microsoft SQL Server database), and click OK. Choose the connection named `pubsConnectionString1` and click Next. Check on the Specify columns from a table, select the name of `authors`, and click the asterisk (all columns). Click the WHERE button and set the `au_id` column equal to a source of `querystring` using the QueryString field `au_id` that you type in. Set a default value of `213-46-8915`, as shown in Figure 1-12. Click Add and double-check that the SQL expression shows `[au-id]=@au_id`. Click OK, click Next, test the query (click OK to accept default values in test and click Finish.

Close the `DetailsView` control's Smart Tasks panel menu, save, and press F5 to run. You will see the details on only one author, and you will see them presented in a `DetailsView` control rather than the GridView of the first part of the exercise. Close your browser.

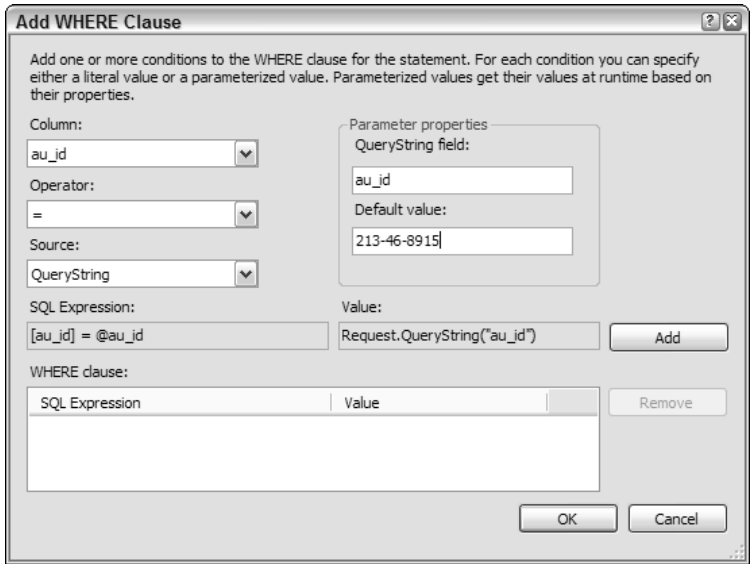

Figure 1-12

16. Now that you have a page to show details about an author, you can finish by giving the user a way to click the GridView of our first page to jump to the details page. In VWD, open `TIO-1-GridViewSql.aspx` in Design view. Select the GridView and expand the small top-right arrow to see the Smart Tasks panel. Select Edit columns to see the Fields dialog box. Select Hyperlink in the Available Fields and click Add. Keep your hyperlink field selected in the lower left, and go to the properties on the right to set the following:

```
Text =  View Details...
DataNavigateUrlFields =  au_id
DataNavigateUrlFormatString = TIO0101-DetailsViewSql.aspx?au_id={0}
```

Click OK and close the Smart Tasks panel.

17. Save the file and then run it. Observe the behavior in the browser.

How It Works

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 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 HTML-like syntax with opening and closing tags, hierarchical tags, and attributes within tags. (We say *HTML-like* because in some cases ASP.NET tags are not 100 percent HTML-compliant.)
- 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 data 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 data source 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 querystring value.
- ASP.NET offers an `AutoPostback` feature that causes the `ViewState` to be sent back to the server and induces the server to process a new page (generally using some change in the page, such as the user's list selection) and send the new page to the browser.

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 — Creating a DataList from Access

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

1. Continue using VWD with the BegAspNet2DB Web site open.
2. Right-click the site's `ch01` folder and Add a New Item using the Web Form template with the name `TIO-0102-DataListAccess.aspx`.
3. In Design view, drag a DataList control from the Data section of the Toolbox to the page. From the Smart Tasks panel, select New Data Source from the data source list. Select the data source type of Access database, accept the default name of `AccessDataSource1`, and click OK.

Choose a database file by browsing to `C:\websites\BegAspNet2Db\App_Data\Northwind.mdb and clicking OK`. Click Next and, for the Name (which means the name 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.

4. You now have a `DataList` control that uses an `AccessDataSource` control that returns the Categories table of Northwind. Test it in your browser and then close the browser before continuing.

5. Back in VWD, select the `DataList`, open its Smart Tasks panel (small arrow at top right) and click the property builder to set some overall properties for the `DataList`. Change the columns to 3 in the `Repeat layout - Columns` property and the direction to horizontal. Click Apply and OK.

6. Now you configure the display of each record's data. Continuing with the `DataList` selected, in its Smart Tasks panel, click Edit Templates. You can now add and format controls within the `ItemTemplate` space.

   First, select and delete the six controls in the template. Check that your insertion bar is in the editable space, and 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 finish all of the dialog boxes and create the table in the `DataList`'s template.

7. Now we will get some data-bound controls into the template. In the top-left cell, we will place the Northwind logo. Prepare by creating an `Images` folder in the root of the site (select the root of the site, right-click, and select New Folder).

   Right-click the new `Images` folder and choose Add Existing Item. Browse to the downloads for this book and select the file named `NorthwindLogo.bmp`. (You can use any other `.bmp` or `.gif` if you don't have the download.) Open the toolbar, find the Standard section, and then drag into the top-left cell of the table an image control (not Image Map or Image Button). Do not use the Image Smart Tasks panel in this case (because we want all records to show the Northwinds logo). Select the image control and, in the Properties window, set the `ImageUrl` to `~\Images\NorthWindLogo.bmp`, as shown in Figure 1-13.

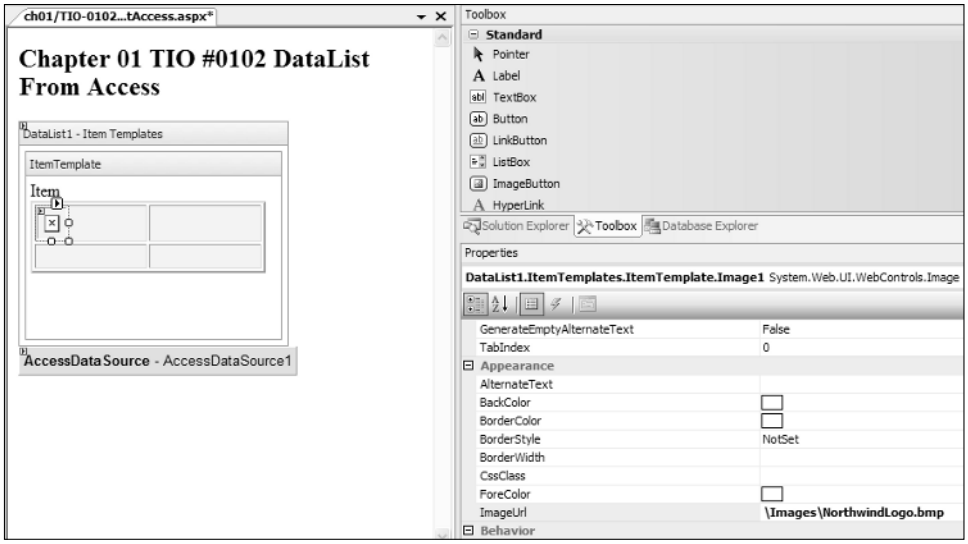

Figure 1-13

8. Into the top-right cell, drag a label from the Toolbox and click Edit Data Bindings in the Smart Tasks panel. Bind its text property to the `category ID` field, as shown in Figure 1-14, and click OK. Into the bottom-left cell, drag a label and bind its text to the description field. Into the bottom-right cell, add a label and bind its text to the category name field.

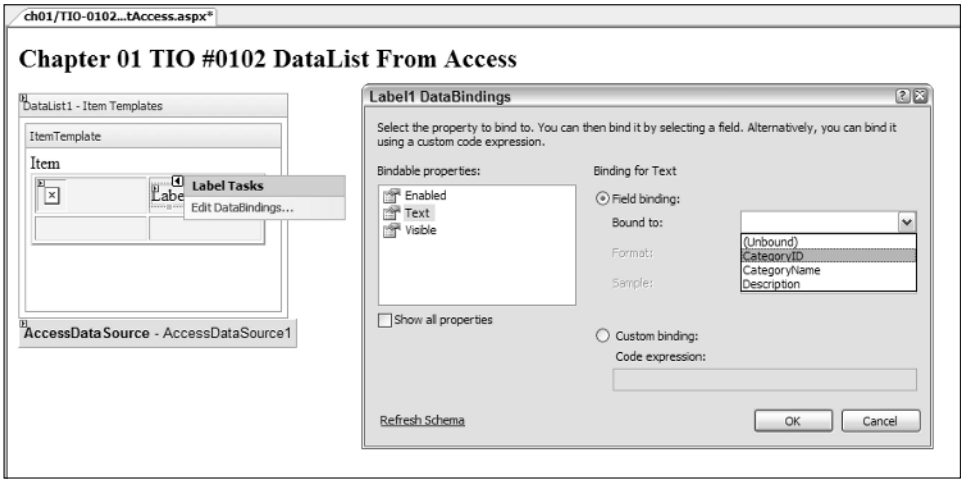

Figure 1-14

9. Now that the cells are filled, select the right column by clicking on the arrow that appears when you position the mouse cursor right at the top of the column. Once selected, in the Properties window, set align to center. Select the left column, and set its width in the Properties window to 600. Finish by opening the `DataList`'s Smart Tasks panel (arrow at top right of entire `DataList` control) and clicking on End Template Editing.

10. Save the document and press Ctrl+F5 to run it. Your page should resemble the following code and display the results in your browser, as shown in Figure 1-15.

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>TIO-0102-DataListAccess.aspx</title>
</head>
<body>
        <h2>Chapter 01 TIO #0102 DataList From Access </h2>
    <form id="form1" runat="server">
    <div>
        <asp:DataList ID="DataList1" runat="server" DataKeyField="CategoryID"
DataSourceID="AccessDataSource1"
            RepeatColumns="3" RepeatDirection="Horizontal">
            <ItemTemplate>
                Item<table border="2">
                    <tr>
                        <td style="background-color: lightgoldenrodyellow"
width="600">
```

```
                            <asp:Image ID="Image1" runat="server"
ImageUrl="~/Images/NorthWindLogo.bmp" /></td>
                        <td align="center" style="width: 100px; background-color:
lightgoldenrodyellow">
                            <asp:Label ID="Label1" runat="server" Text='<%#
Eval("CategoryID") %>'></asp:Label></td>
                    </tr>
                    <tr>
                        <td style="background-color: lightgoldenrodyellow"
width="600">
                            <asp:Label ID="Label2" runat="server" Text='<%#
Eval("Description") %>'></asp:Label></td>
                        <td align="center" style="width: 100px; background-color:
lightgoldenrodyellow">
                            <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="~/App_Data/Northwind.mdb"
            SelectCommand="SELECT [CategoryID], [CategoryName], [Description] FROM
[Categories]">
        </asp:AccessDataSource>

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

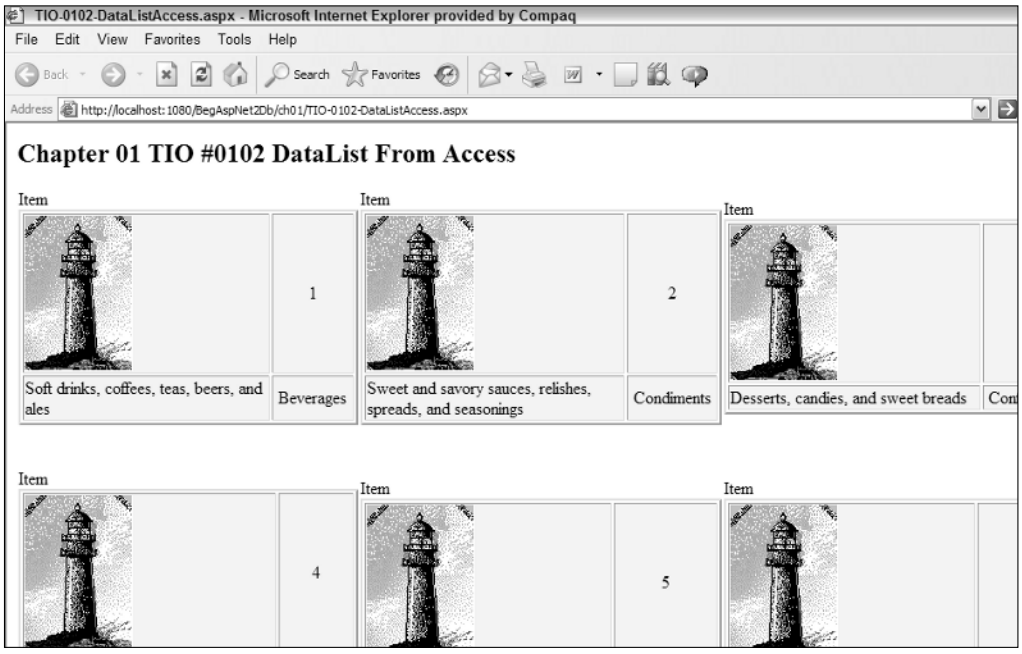

Figure 1-15

How It Works

You can see the same patterns here as in the first exercise. Instead of writing code, you use VWD drag-and-drop to put ASP.NET 2.0 data controls onto the page. You can build a page in just a few minutes. As before, you added a data-bound control (`DataList`) and it automatically walked you through setting up its data source control (`AccessDataSource`).

The `DataList` control starts the same way as the `GridView`, by specifying its `DataSource` control. Then we revised the rendering through the control's item template. In this template, we added and formatted controls and bound them to fields in the underlying data source control. The template 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 — Using a TreeView Based on XML Data

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 will 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. Select your site's `App_Data` folder (created by VWD), right-click, and select Add Existing Item. Browse to the downloads for this book and select the file named `Bookstore.XML` from the downloads into your `App_Data` folder. Repeat the process to get `closedbook.gif`, `notepad.gif`, and `folder.gif` into your `Images` folder (feel free to import all the images of the download, we will use them throughout the book). In your `ch01` folder, create a new Web Form page named `TIO-0103-TreeViewXML.aspx`.

2. Start by adding a data source control as follows. 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 need to fill in only two items (double-check your syntax and case). For the Data File, browse to `\App_Data\Bookstore.xml`. For the `XPath` expression, type the following with careful attention to case.

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

3. Now we need to display the data in a data-bound control, so drag a `TreeView` control from the Navigation 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).

4. Save and run the page at this point to see, as shown in Figure 1-16, a tree of the generic names of the XML nodes (book, chapter), but not the actual values (Tale of Two Cities, Introduction).

5. Exit your browser and go back to the page in Design view. Open the Smart Tasks panel menu for the `TreeView` and click `EditTreeNode DataBinding`. In the available list, click `Book` and click Add. Set three properties for book, as follows. For the `ImageUrl`, use the ellipses icon to browse to the `closedbook.gif` in your site's `Images` folder.

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

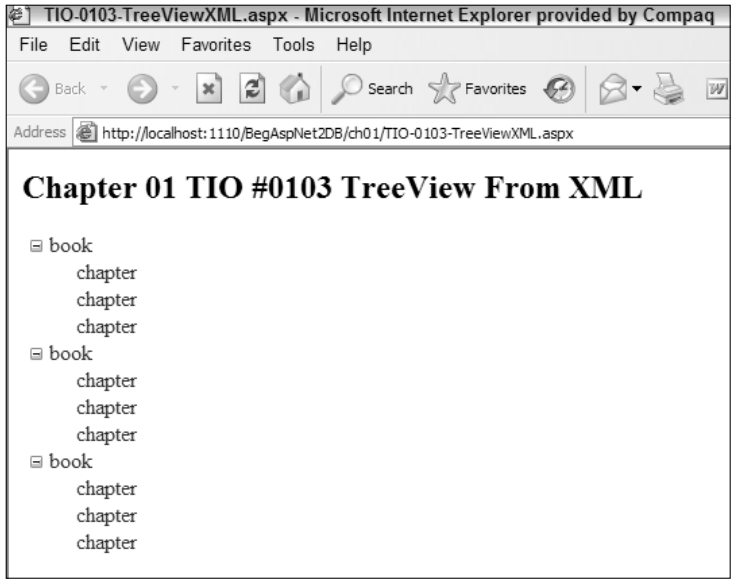

Figure 1-16

6. Staying in the DataBindings dialog box, select the chapter node in the "Available data bindings" and click Add. Then set its `DataMember` to chapter, `TextField` to name, and `ImageUrl` to `notepad.gif`. Click Apply and OK to close.

7. Save the page and view it in your browser, as shown in Figure 1-17.

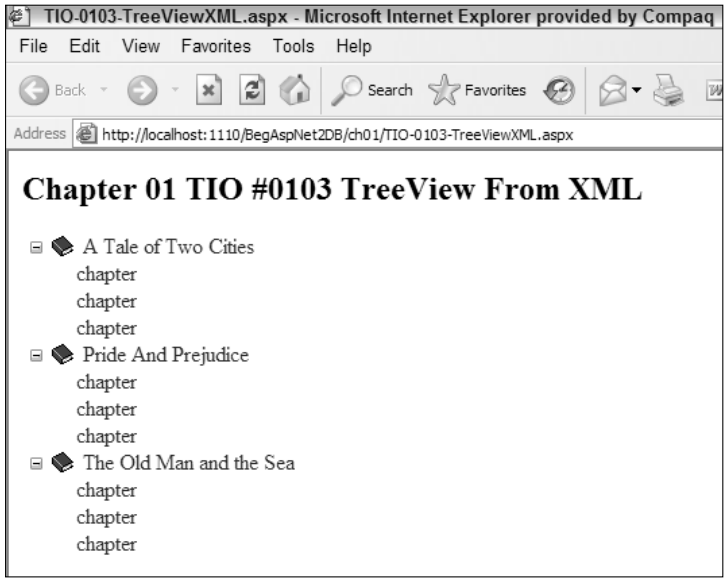

Figure 1-17

8. You will add one last feature to demonstrate that you can handle events in a similar fashion to older versions of ASP.NET. Exit your browser and go to VWD. In Design view, select the `TreeView` control. Display the Properties window by pressing F4. At the top of the Properties window, click the lightning bolt icon and then double-click the `SelectedNodeChanged` event. VWD will automatically switch to source view and type the first and last lines of an event handler. You can now type in the highlighted line below. Don't test the code in the browser until after the next step, wherein we add a short bit of code. The first option uses Visual Basic, and we demonstrate C# following that.

VB

```
<%@ 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>
```

The second option employs C#, as follows, if you chose to use C# when you created the page. Note both a change in the code and in the initial `Page Language` setting.

C#

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

9. Take a look at the line added automatically by VWD that instructs the `TreeView` control to use your new event handler when any value of the tree is clicked. Staying in Source view, scroll down to and then observe the following highlighted line in the `<asp:TreeView>` control:

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

10. 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:

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>
...
```

VB

```
<%@ 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//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1.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>
```

How It Works

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 actual 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 continue to work. Also, 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.

The last exercise ended with a little coding. ASP.NET supports many languages, but the most common are Visual Basic and C#. We cover both in this book. The language is set in a directive at the top of the page, which VWD automatically creates based on your selection when creating the page. As in earlier versions of ASP.NET, you can set an event to call 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`.

Common Mistakes

Near the end of each chapter, we provide a list of some common mistakes that beginning programmers make with the material covered in that chapter. Of course there are lots of ways to go wrong, but these are the ones that come up most often when we teach, observe code, or answer questions. If you are having difficulty getting a page to work, we suggest you run through this list of mistakes to help guide your troubleshooting. In addition, reading through these scenarios will serve as a review of the chapter's concepts.

- Modifying the wrong data source control when more than one data source control is on a page
- Attempting to use an Access MDB file that has password protection (see Chapter 4 for information on using secured MDB files)

Summary

This chapter reviewed some basic ASP.NET 2.0 topics, covered setting up your machine for the book, and walked you through three exercises that displayed data.

Recall from your study of ASP.NET 2.0 that pages are stored as a combination of text, HTML tags, scripts, and ASP.NET 2.0 server-side controls. When a page is requested, IIS directs the request to ASP.NET 2.0, which processes the page using the ASP.NET 2.0 controls and code 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 HTML.

The .NET Framework is a collection of classes written by Microsoft that enable programmers to quickly and easily create solutions for their organization. ASP.NET, a set of classes in the namespace `System.Web`, provides very powerful controls for building dynamic Web pages. ASP.NET 2.0 greatly reduces the amount of coding for common data scenarios, compared to earlier versions of ASP. In most cases, the decisions about ADO.NET 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: Public deployment requires IIS, whereas development can use the lighter-weight ASP.NET Development Server that comes with Visual Web Developer. To create pages, we use Visual Web Developer Express because it offers many tools in its IDE and is available for free. Several alternatives for a database management system were discussed also. This book uses SQL Server Express (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 copied to your site two databases (`Northwind.mdf` and `Pubs.mdf`) 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 in the first Try It Out 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 examine data source controls. Topics include Access, SQL data source control, and connecting to other databases. Following that, several chapters 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 source controls that ship with ASP.NET 2.0.
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. Compare SQL Server, MSDE, and SSE, and make some observations.