

Microsoft ASP.NET Professional Projects

by Hersh Bhasin <u>Premier Press</u> © 2002 (638 pages) ISBN: 1931841217

Teaches Web developers how to build powerful applications using the .NET Framework and Microsoft's ASP.NET.

Table of Contents

Microsoft ASP.NET Professional Projects	
Part I - The ASP.NET Programming Environment	
Chapter 1 - Introducing ASP.NET	
<u>Chapter 2</u> - Introducing ASP.NET Web Forms and Controls	
Chapter 3 - Using ADO.NET in the .NET Framework	
Chapter 4 - Data Binding	
Chapter 5 - Input Validation	
Chapter 6 - User Controls	
<u>Chapter 7</u> - Custom Controls	
Chapter 8 - Business Objects	
Chapter 9 - Working with ASP.NET Web Services	
Chapter 10 - ASP.NET Applications	
<u>Chapter 11</u> - Caching	
Chapter 12 - Tracing	
Chapter 13 - Security	
Part II - Projects	
Project 1 - A Personal Finance Manager	
<u>Chapter 14</u> - The Design of the Personal Finance Manager	
Chapter 15 - Chart of Accounts	
Chapter 16 - Transactions	
Chapter 17 - The Trial Balance Report	
Project 2 - Web Services	
Chapter 18 - Creating a Generic Database Web Service	
Chapter 19 - Designing a Navigation System	
$\underline{Chapter \ 20} \ \text{-} \ \text{Incorporating Web Services in the Chart of Accounts Form}$	٦
<u>Chapter 21</u> - Incorporating Web Services in the Transactions Form	
Chapter 22 - Incorporating Web Services in the Trial Balance	
Project 3 - Inventory Management System	
Chapter 23 - The Design of the Inventory Management System	
Chapter 24 - Inventory Masters	
<u>Chapter 25</u> - Inventory Movements	
Chapter 26 - The Inventory Balances Report	
Project 4 - The GenEditAdd Control	
Chapter 27 - Using the GenEditAdd Control	
Chapter 28 - Extending the GenEditAdd Control	
Project 5 - Visual Studio.NET	
$\underline{Chapter \ 29} \ \text{-} \ \text{Displaying Database Data Using a Strongly-Typed DataSet}$	۶t
Chapter 30 - Writing CRUD Applications with Visual Studio.NET	
Chapter 31 - Creating a Web Service Using Visual Studio.NET	
Part III - Appendixes	
Appendix A - Installing the Sample Database	
Appendix B - HailStorm	

Index List of Figures List of Tables List of Examples

Microsoft ASP.NET Professional Projects Hersh Bhasin

© 2002 by Premier Press, Inc. All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system without written permission from Premier Press, except for the inclusion of brief quotations in a review.

The Premier Press logo, top edge printing, related trade dress and Professional Projects are trademarks of Premier Press, Inc. and may not be used without written permission. All other trademarks are the property of their respective owners. *Important:* Premier Press cannot provide software support. Please contact the appropriate software manufacturer's technical support line or Web site for assistance.

Premier Press and the author have attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer.

Information contained in this book has been obtained by Premier Press from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, Premier Press, or others, the Publisher does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from use of such information. Readers should be particularly aware of the fact that the Internet is an ever-changing entity. Some facts may have changed since this book went to press.

ISBN: 1-931841-21-7 Library of Congress Catalog Card Number: 2001096478

Printed in the United States of America 02 03 04 05 06 RI 10 9 8 7 6 5 4 3 2 1 Publisher Stacy L. Hiquet Associate Marketing Manager Heather Buzzingham Managing Editor Sandy Doell Acquisitions Editor Kevin Harreld Editorial Assistant Margaret Bauer Technical Reviewer Mingyong Yang Copy Editor Jenny Davidson Interior Layout Marian Hartsough Associates Cover Design Phil Velikan Indexer Kelly Talbot Proofreader Kim Cofer Dedication

To my parents, my wife Ritu, and my daughter Ria **Acknowledgments**

I thank my wife Ritu for motivating me to write this book and for painstakingly proofreading, editing, and formatting all my manuscripts. I thank all my friends at Premier Publishing who made this book possible. Thank you Kevin Harreld and Jody Kennen for putting your trust in me, Elizabeth Agostinelli, Jenny Davidson and Brian Thompson for wading through my work and fixing what was wrong and Mingyong Yang for reviewing my source code and giving valuable suggestions.

About the Author

Hersh Bhasin has been consulting on Microsoft technologies for some nine odd years and maintains a Web site on emerging technologies like .NET, SOAP, XML at <u>http://hersh.weblogs.com.</u> He qualified as a Management Accountant from The Chartered Institute of Management Accountants - UK (CIMA) and also obtained a Bachelor of Science degree from the University of Punjab, India. He can be contacted at <u>hersh_b@yahoo.com</u>

Part I: The ASP.NET Programming Environment

Chapter List

Chapter 1: Introducing ASP.NET Chapter 2: Introducing ASP.NET Web Forms and Controls Chapter 3: Using ADO.NET in the .NET Framework Chapter 4: Data Binding Chapter 5: Input Validation Chapter 6: User Controls Chapter 7: Custom Controls Chapter 8: Business Objects Chapter 9: Working with ASP.NET Web Services Chapter 10: ASP.NET Applications Chapter 11: Caching Chapter 12: Tracing Chapter 13: Security

Chapter 1: Introducing ASP.NET

Overview

ASP.NET is a radical evolution of ASP and its associated data access service, ADO, which is now called ADO.NET. ASP suffered from many limitations—it was unstructured, so the code intermingled with the presentation logic, which made applications difficult to understand and maintain. Due to this limitation of ASP, code segregation was not possible. You could not hand over the presentation logic to a Web designer and the code to a developer and ask both to work simultaneously on the application. Unlike windows-based application development, ASP did not have an inherent component or programming model. ASP developers had to use a combination of markup languages, scripting environments, and server platforms to get their work done. Tool support was limited and although Visual InterDev introduced a Visual Basic type interface that allowed you to drag and drop components such as text boxes and labels onto a form, it was clunky and added tons of code to the form, which needless to say scared away most developers from ever using this product.

ADO, the Data Access component of ASP, had been designed with a view to serving the data access needs of client/server-based applications. Programming for the Web, however, followed different rules. A client/server application had no need to optimize database connections and a typical database operation would open a database connection and leave it open until the looping operation of an ADO recordset was complete. Database connections in a Web-based environment, however, were expensive. Web programming required a disconnected way of manipulating data. Thus the Remote Data Services (RDS) were born. With the advent of XML (*eXtensible Markup Language*), the request/response paradigm became the order of the day. To keep up with this message-based system of communication, HTTP support was added to RDS, which allowed business logic to be called at the middle tier. XML follows a

heterogeneous and hierarchical data model (XMLDOM) whereas MDAC (*Microsoft Data Access Technologies*) follows a relational model. To work with XML data we had to make a choice between MSXML and MDAC. But ADO.NET solves this dilemma. XML support is built at a very basic level and it is quite similar to working with "database" data. No longer is choosing between MDAC and MSXML an issue.

Web forms, which will be discussed in <u>Chapter 2</u>, "<u>Introducing ASP.NET Web Forms and</u> <u>Controls</u>," are the fundamental building blocks of ASP.NET. The concept of "Code Behind" has been introduced, which is the process of writing presentation logic and script in separate files. Code Behind seeks to eliminate the clutter and "spaghetti" code (spaghetti code is code where the scripting portion intermingles with the presentation logic) that traditional ASP seemed to encourage. ASP.NET provides a server-based, event-driven programming model similar to Visual Basic, which enables WYSIWYG tools like Visual Studio to be used.

ASP.NET introduces two sets of controls, the HTML controls and the Web controls, which are collectively known as "<u>server controls.</u>" These controls render HTML for Web browsers in addition to providing means of preserving state across round trips, detecting the browser types (and rendering HTML accordingly), and serving as the building blocks for composite controls. These controls reside on the server and output plain HTML to the browser. Since all browsers can understand HTML, they are able to overcome the classic cross-browser compatibility problem.

The HTML controls (textbox, form, button, and so on) are the normal HTML controls that we have been using so far, with a new runat="server" attribute added. The sole use of these controls is to provide a quick migration path to ASP.NET as any HTML control can be converted to an ASP.NET control by adding the runat="server" attribute. The Web controls, however, provide a high degree of abstraction and usefulness. Four types of Web controls exist: Intrinsic controls, Rich controls, List Bound controls, and Validation controls. Intrinsic controls are the ASP.NET versions of traditional HTML controls, such as textboxes, buttons, and DropDownList. These controls have a special prefix of ASP.NET that distinguishes them from the normal HTML controls and they also have a runat ="server" attribute. Thus a textbox is created as follows:

<asp:TextBox id="Text1" runat="server"/>

The purpose of Web controls is to simplify the nomenclature of a control. Controls that overlapped in their functionality have been reduced to a single control. Properties like ForeColor, Font, BackColor, Enabled, and so on are consistent among controls. The developer thus needs to remember one property setting that he can apply to all controls.

Rich controls consist of the Calendar and AdRotator. The Calendar outputs HTML for downlevel browsers (these are browsers that do not support DHTML) or DHTML for uplevel browsers. The AdRotator displays rotating advertisements.

List bound controls are the subject matter of <u>Chapter 4</u>, "<u>Data Binding.</u>" There are three controls in this category: the DataGrid, the DataList and the DataRepeater. These controls automate the task of displaying database data as lists and data grids. The developer applies a number of templates to these controls to achieve a high degree of customization. The DataGrid can even be used for in-place editing of data. Validation controls, which are discussed in <u>Chapter 5</u>, "<u>Input Validation.</u>" automate the

mundane activity of writing validation code. There are five validation controls and one validation summary control. The validation controls are the

RequiredFieldValidator, RegularExpressionValidator,

CompareValidator, RangeValidator, CustomValidator, and the

ValidationSummary. The work of each of these controls is evident from its name. For example, the RequiredFieldValidator does not allow the user to leave the required field blank. Similarly, the RangeValidator verifies whether user input falls within a specified range. It is a simple task to incorporate validation in an ASP.NET web form. All you need to do is associate an input text box with the appropriate validation control. ADO.NET, the latest avatar of ADO, is discussed in <u>Chapter 3</u>, "<u>Using ADO.NET in the</u> <u>.NET Framework.</u>" ADO has seen a massive overhaul (a complete rewrite would be a better choice of words) in ADO.NET. The foundation of ADO—the recordset—has been given the golden handshake. As noted above, the recordset understood only the

relational way of doing things, which was appropriate for handling database data. With the advent of XML, which followed a heterogeneous and hierarchical data model, the recordset had a hard time keeping up. A new object called the DataSet has been introduced in ASP.NET. The DataSet is an in-memory copy of the database, complete with tables, columns, relationships, constraints, and data. It allows relationships to exist between multiple tables, analogous to a foreign-key relationship in a database. You can navigate between tables based upon their relationships. The DataSet has some outstanding qualities. For example, it can talk to a variety of datasources; it can receive data from a database, an XML file, code, or user input. No matter what the source of the data within the DataSet is, it is manipulated through the same set of standard APIs. The DataSet is transmitted as an XML stream and can thus be passed between any objects (not just COM objects) without interference from firewalls. To transmit an ADO disconnected recordset from one component to another, COM marshalling had to be used.

User controls, which are discussed in <u>Chapter 6</u>, "<u>User Controls</u>," are the evolution of the server-side include files. Include files are static files. User controls, on the other hand, provide object model support, which enables you to program against properties and use methods. User controls work much like the ASP intrinsic controls in that they can expose properties and methods. In <u>Chapter 6</u>, I design a user control that automates building of the navigation links for a Web site based on the URLs specified in an XML file.

ASP.NET has a very clean and elegant approach to authoring custom controls. In <u>Chapter 7</u>, "<u>Custom Controls.</u>" I discuss the process of authoring custom controls in detail. I also show you how to build a component (which I call "GenEditAdd") that you can use to extend the DataGrid's functionality. The DataGrid does not have the functionality to insert records. Using the GenEditAdd component, you can automate the process of record insertion. You can also use the GenEditAdd component in lieu of the editing functionality provided by the DataGrid, which requires you to code a number of events. The GenEditAdd component requires simple property settings and the code generation is automatic.

Encapsulating business logic into components has always been an important part of both Web and client/server applications. ASP.NET has greatly simplified the process of registering components. If you have developed COM objects in the past, you must know the pain involved in registering components. A component had to be registered using the regsvr32 utility. If the component was modified, the entire Web server had to be stopped in order to re-register the component. This was called "DLL Hell." In ASP.NET. components are simply copied and pasted in the bin directory and no registry updates are required. Chapter 8, "Business Objects," looks at this important topic. Web service is the main protagonist of the .NET arena and the content of this book and its projects reflect its importance. A web service is a combination of three things: a function written in a .NET-compliant language, SOAP, and XML. When you need to reuse logic in a number of places, the best way to do so is to write the code as a function. A collection of functions that share some common goal can be combined into a business object. For example, the four basic database operations are insert, delete, update, and select. We can write a generic function for each operation and pack them together in a business object called (say) DataBaseClass. Now this class, together with its functions, can be initiated and called in any object that needs to use its functionality. A web service is a Web-enabled business object, which is a collection of functions that can be called over the Web. Functions written for a web service are written as normal functions, and the only difference is that the functions are preceded with a special tag that marks them as web services. A standard called SOAP (Simple Object Access Protocol) sets out the rules that must be followed by the machine that makes a function call and the machine that responds to that call by sending a resultset back. The request and response is made in XML and the XML document follows the rules set out in the SOAP standard. Exchanging information as XML enables a client application to call a function on the server, regardless of what operating system each is running, what programming language each is written in, or what component model is supported on each. This is because XML is basically a text file that all machines can understand and because SOAP uses HTTP, which is the most common Internet transfer protocol and

one that is used by essentially all Web browsers. <u>Chapter 9</u> "<u>Working with ASP.NET</u> <u>Web Services.</u>" provides a detailed discussion on web services.

<u>Chapter 10, "ASP.NET Applications.</u>" covers ASP.NET applications. An ASP.NET application is an IIS virtual directory and its subdirectories. All of your Web application files go into this folder. This folder has a special subdirectory called bin. All the compiled business objects and web services reside here. When you want to register a new component, you just copy and paste the DLL file in this folder (as opposed to using regsvr32). This folder also contains two special files: web.config and global.asax. The web.config file is an XML file that you use to configure various facets of the application. For example, you can use it to set up and configure security, caching, or tracing. The global.asax file contains application-level program directives, handlers for application and session-level events, and declarations of objects that are globally accessible to all parts of the application. In general, this file enhances the functionality that was provided by the global.asa file in ASP.

<u>Chapter 11</u>, "<u>Caching</u>," deals with caching, which is the process of keeping frequently visited Web pages in memory. The theory behind caching is that there are some items of the Web site that are very expensive to construct and that such items should be created once and then stashed away in memory for a fixed duration of time or until they change. Subsequent calls to these resources will not re-create the resource but simply retrieve it from the cache. Such items are typically resources that remain unchanged over a period of time; for example, shopping lists or price lists.

<u>Chapter 12</u>, "<u>Tracing</u>," discusses tracing. Developers have often resorted to writing a number of Response.Write() statements in the code to try to debug errant code. When the problem is located, these debugging statements must be cleared out. This method is cumbersome and error-prone, because you might accidentally remove code along with the debugging statements. ASP.NET introduces an elgant way of writing such debugging code. Debugging is enabled by adding a page-level directive (or by enabling it in the web.config file). Debugging statements are then written using Trace.write() instead of Response.Write(). When the form has been debugged, there is no need to remove these statements from the body of the form. You can simply disable Trace and these statements will not be displayed in the browser.

Security is discussed in <u>Chapter 13</u>. ASP.NET implements authentication through authentication providers. These authentication providers are modules that contain code required to authenticate the credentials of the requesting user. Three authentication providers are currently available: Windows Authentication, Passport Authentication, and Cookie Authentication. All three providers are discussed.

In <u>Project 1</u> (Chapters 14 to 17), I show you how to build a *Web-enabled* personal finance manager using ASP.NET web forms. This project is spread over four chapters. In this project, I take a product that has its roots in the client/server era—a personal finance accounting module—and revamp it for the Web. A personal finance manager is an accounting application, such as Quicken or Microsoft Money that enables you to maintain bank, cash, credit cards, and investment accounts. This project is designed to be a production quality accounting application and makes use of stored procedures and database triggers. It's comprised of web forms to maintain your chart of accounts, transactions details and it even draws up a trial balance report.

The Internet brings some exciting possibilities to the traditional way of designing applications. The various modules of an accounting application need no longer be connected with "wire." Using ASP.NET and web services, we can design applications that can send and receive data using the Internet and HTTP. In <u>Project 2</u> (which spreads over five chapters), I build generic database access services that can then be used to interact with any database. This service has functionality to insert, update, delete, and select records from a database. This web service accepts a database connection and a valid SQL query as parameters. If the query is an action query (insert, update, or delete), the appropriate action is performed. If the query is a select query (which returns a resultset), a DataSet is returned to the calling object. This DataSet can then be used to bind a control like a DataGrid. I demonstrate this service by incorporating it in the personal finance manager that was developed in <u>Project 1</u>. This project also demonstrates use of the navigation user control that was built in <u>Chapter 6</u>, "<u>User Controls.</u>" This navigation control builds the site navigation of the application using URLs defined in an XML file.

The advantage of having a navigation system separate from the main application is that you can add or delete links (by modifying the XML file) without having to change the Web pages in the application.

In <u>Project 3</u> (Chapters 23 to 26), I have taken another application that has traditionally been a client/server application and revamped it for the Web. This is an inventory management application. This application makes use of the database web service class that was developed in <u>Project 2</u>. It also makes use of various stored procedures and triggers.

In <u>Project 4</u> (<u>Chapters 27</u> and <u>28</u>), I enhance the functionality of the custom control GenEditAdd (which was initially developed in Chapter <u>7</u>). The GenEditAdd control can be used to insert or update database records. The DataGrid does not have the capability to insert records, although it does have editing capabilities. The edit mode of the DataGrid is quite cumbersome, as you have to code a number of events in the DataGrid for the process to work. This control was developed to enhance the usefulness of the DataGrid. It can be hooked up to a DataGrid to provide both editing and insertion capabilities in a consistent manner. This control works by setting various properties and the code generation is automated.

In <u>Project 5</u> (Chapters 29 to 31), I discuss the important features of Visual Studio. In <u>Chapter 29</u>, "<u>Displaying Database Data Using a Strongly-Typed DataSet</u>," I begin with an overview of the important features of Visual Studio.NET, focusing on the various wizards, tools and components available. I'll also show you how to use the typed DataSet to display database information using the Visual Studio.NET drag and drop features. In <u>Chapter 30</u>, "<u>Writing CRUD Applications with Visual Studio.NET</u>," I'll show you how to interact with the database using Visual Studio.NET. I'll show you how to add, delete, and update database rows. I'll also show you how to customize a DataGrid by enabling paging and sorting from within the Visual Studio.NET. Finally, in <u>Chapter 31</u>, "<u>Creating a Web Service Using Visual Studio.NET</u>," I'll show you how to develop and consume web services with Visual Studio.NET.

Installing the .NET Framework SDK

The .NET SDK can be downloaded from the Microsoft download site at <u>http://msdn.microsoft.com/net/.</u> It is quite a large download and you might want to consider ordering a CD, which will ship at a nominal charge.

There are two versions available; a standard version or a premium version. The premium version includes additional features like output caching, web farm session state, code access hosting and support for four and above CPUs.

Installation is straightforward and involves running the setup.exe. If prompted, you should update the Windows Installer Components. You should also apply the latest patches for your Windows version. You should also update your version of MDAC (*Microsoft Data Access Components*) to the latest version, which is currently version 2.7. If the installer complains that ADO 2.7 is not installed, you can still proceed with the installation by disregarding the complaint. You will be given a choice to install the SDK samples. The samples are a rich source of information and you should choose to install them. A named instance of the Microsoft Data Engine (MSDE) is installed along with the samples and this contains the sample database.

Тір

A limited-time evaluation copy of Microsoft SQL Server can be obtained from

http://www.microsoft.com/sql/evaluation/trial/2000/default.asp. You

can also order this copy on a CD and only pay the cost of shipping. After you have SQL Server up and running, install the ASP.NET QuickStart samples. These samples are an excellent training resource on ASP.NET. To install these samples, open the Microsoft NET Framework SDK/Samples and QuickStart Tutorials link, which is added to your programs during the SDK installation and follow the installation steps. Once the samples are installed, they can be accessed at http://localhost/quickstart/default.aspx. After you install the SDK, all you need is a text editor to write your scripts. You can also order the Visual Studio CD set (again at a nominal charge) and use it to develop your scripts. If you have the Visual Studio CDs, the Framework SDK is on the second CD. I have discussed development with Visual Studio, where appropriate, and one entire project (Project 5) is dedicated to exploring this development tool. I have left discussion of Visual Studio till the end because I want my readers to be familiar with the internals of ASP.NET before using the wizard-like tools of the Visual Studio IDE, which hides the intricacies of code development. A text editor that

I highly recommend is TextPad, which is shareware and available at <u>http://www.textpad.com</u>. You can also download the syntax definition file for .NET from its site. This file will display various ASP.NET keywords in different colors.

Chapter 2: Introducing ASP.NET Web Forms and Controls

ASP.NET forms are designed to overcome a number of shortcomings inherent in ASP pages. In these pages the HTML elements and script code are necessarily intertwined making the resultant page very cluttered. These pages are not easily edited with WYSIWYG tools. ASP.NET improves on the ASP page and adds many interesting enhancements to it. It provides a server-based, event-based programming model similar to Visual Basic. It introduces a technique called "Code Behind," which allows the developer to keep the script code in a file separate from the HTML markup. ASP.NET introduces two sets of controls, the HTML controls and the Web Controls, which are collectively known as "server controls." These controls render HTML for Web browsers in addition to providing means of preserving state across round trips, detecting the browser types (and rendering HTML accordingly), and serving as the building blocks for composite controls. A round trip occurs whenever the user submits a form or an event occurs that causes a post to the server; for example, the user fills out a text box on a form and clicks on the submit button. The server processes the information passed onto it and sends the page back to the client for display. The original state of the form is maintained by ASP.NET. This means that when the user fills out a text box and submits the form to the server, the text box will retain this information even after the round trip. This is a welcome change from traditional ASP programming where the developer had to take care of maintaining state, as the user-input values were lost after every post.

Basic Techniques

To create an ASP.NET form you simply save a text or HTML file with the .aspx extension. No special tools are needed and you can use an editor like Notepad for the job. You can also use Visual Studio.NET, a rapid application development environment (RAD) that allows you to drag and drop controls onto a form.

ASP.NET forms also provide selective backward compatibility. For example, you can use normal ASP code and mix script tags with HTML elements using the <% %> blocks. ASP and ASP.NET applications can run side by side on the IIS without interference. However ASP applications developed using the Visual Basic Scripting Edition will need to be modified to port to ASP.NET.

In ASP.NET, script blocks are compiled and not interpreted, leading to enhanced performance. Compiling the code involves converting the code instructions to the machine language. In ASP.NET however, code is not compiled to machine language directly. It is instead compiled to an intermediate language called Microsoft Intermediate Language (MSIL or IL). IL code is further compiled to machine language using the JIT compiler (just-in-time compiler). The JIT compiler compiles each portion of code as it is called, instead of compiling the complete application in one go. This leads to faster start-up time. The resultant compiled code is stored till the application exits and hence does not have to be recompiled each time that portion of code gets called. Using this process, it is expected that execution of IL code will be almost as fast as executing native machine code.

State Management

Though you can use the script blocks, they do not lead themselves to a clean programming environment. Consider the basic requirement of maintaining state in a "post back" form. This is a form that accepts user input and "posts back" to itself. It needs to remember the values entered so that, if the user makes a mistake, it can display the values the user had earlier entered so that he can correct them. Coding for such a form in the ASP environment has involved using the Response object to extract the value and a <% =some variable %> block to display the passed value. Here is an example:

State.asp

<html>

<form method="post">

<h3> Name: <input name="Name" type=text value="<%=Request.form("Name")%>">

```
<br> <input type = "submit" value = "Submit">
```

</form>

</html>

In ASP.NET, state management is enabled automatically when you use server controls within a form control as follows:

State.aspx

<html>

```
<body style="background-color='beige'; font-family='verdana'; font-size='10pt">
```

<form method="post" runat=server>

<h3> Name: <asp:textbox id="Name" runat="server"/>

```
<asp:button text="Lookup" runat="server"/>
```

</form>

</body></html>

Note that the form automatically "remembers" input values. There are a few drawbacks associated with using server controls for state management. You can only use the "post" method and can only have a single form on your page.

Page Events

ASP.NET has object orientation at its heart. You can code various events in a Visual Basic–like manner. As the form loads, the Page_Load event is fired, form controls become available for use and, as the user continues to interact with the form, other events are generated. The form unload event occurs when the page is unloaded. Due to this event-based structure, a developer can finally start applying event-based coding techniques to Web applications. Figure 2.1 shows you how to accept user-input values to perform a calculation using these techniques.



<%@ Page Language="vb" %>

<html>

Events.aspx

<head>

<script Runat="server">

Sub Calculate(src As Object, e As EventArgs)

Amount.Text = Cstr(cint(qty.text)*cint(price.text))

End sub

</script>

</head>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<h4> Page Events </h4>

<form method="POST" runat="server">

Qty:<asp:TextBox id="Qty" Runat="server"/>

Price: <asp:TextBox id="Price" Runat="server"/>

Amount:<asp:TextBox id="Amount" ReadOnly = "true" Runat="server"/>

<asp:Button id="btnCalculate" Text="Calculate" OnClick="Calculate" Runat="server"/>

</form>

</body>

</html>

At the top of the page, I specify that we are going to be using Visual Basic as the scripting language with the @ Page Language declaration. Each object can be assigned an id property. This enables me to extract property values for the object using its id property. In this example, I am accessing the text property for the Price and Qty textboxes instead of accessing the posted data using the response object. I put my code in the OnClick event of the button, thus making use of the new event-based paradigm of ASP.NET. Finally, note how I am able to set the ReadOnly property of the Amount textbox simply by setting its property value to "true".

Code Behind

As mentioned earlier, a major limitation of ASP is the way the script code intermingles with the HTML tags. This makes the separation of content from presentation difficult. The page becomes difficult to maintain and, in shops where developers and designers work together, segregation of tasks becomes impossible.

Code Behind is a technique to separate the content from the script. A form can become really cluttered with script code and html tags. To reduce this clutter you can lift out all the script code from a web form and put it in a separate file. If you are using Visual Basic code, this file will have an extension of .vb and if you are using C#, .cs.

The first thing you do in a Code Behind file is to import namespaces. This is the first construct in the Code Behind file. *Namespaces* can be thought of as including references in a Visual Basic project. When you make a reference to a DLL in Visual Basic, you can access the methods contained in that DLL. Similarly, by importing a namespace, you can access all the functionality residing within that namespace in your web form. If you do not use this declaration, you have to provide a fully qualified path when referring to a method residing in a particular namespace. This fully qualified path name can become very long (and a pain to type). Using the import directive allows you to directly refer to the method by name. Here are some of the commonly used Namespaces:

- The System namespace contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.
- The System.Collection namespace contains classes that define lists, queues, arrays, hashtables and dictionaries.
- The System.Web.UI.Control is the parent of all Web Form Controls. Three commonly used controls belong to this namespace—Page, UserControl and LiteralControl. Every ASP.NET page is compiled to the Page control by the ASP.NET page framework.
- The System.Web.UI.WebControl namespace contains classes that define the ASP.NET server controls.

- The System.Web.UI.HTMLControls namespace contains classes that define HTML controls.
- Namespaces like System.Data, System.Data.OleDb, System.Data.
 SqlClient, System.XML are classes that deal with manipulating database, XML and other data. I will look at these namespaces in <u>Chapter 3</u>.

I will be discussing these namespaces at various places in the book. In this chapter, I will be discussing the **System.Web.UI.WebControl** namespace and the

System.Web.UI.HTMLControls namespace.

Note

"Imports" is a Visual Basic construct. If you are using C#, you will substitute "Using" for "Imports".

If you have included Web Controls in your .aspx form and want to refer to them in your Code Behind file your import construct will look like the following:

Imports System

Imports System.Collections

Imports System.Web.UI

Imports System.Web.UI.WebControls

You then define a class. All your functions and subs go in this class. A Visual Basic Code Behind file might look like this:

Imports System.Data

Public Class BaseClass

Inherits System.Web.UI.Page

Sub somesub()

End Sub

Function somefunction()

End Function

End Class

Note that I have introduced the Inherits keyword here. The difference between the Inherits and Imports keyword is that the Imports statement only brings in the definition of a set of functionality, but does not actually make use of it. It is comparable to including a reference in Visual Basic. The Inherits keyword is more dynamic. An object that inherits certain functionality can also override and/or extend the parent functionality.

This form becomes a base class and one that your aspx page will inherit from. Inheritance is quite simple. You just have to put a statement at the top of the aspx form: <%@Page Language="VB" Inherits="BaseClass" Src="nameofCodeBehind.vb" %>

Let's take a look at an example. We will take the Events.aspx file and split it into two files: events_cb.aspx and events_cb.vb, the Code Behind file.

events_cb.aspx

<%@Page Language="VB" Inherits="BaseClass" Src="events_cb.vb" %>

<html>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<form method="POST" runat="server">

Qty:<asp:TextBox id="Qty" Runat="server"/>

Price: <asp:TextBox id="Price" Runat="server"/>

Amount:<asp:TextBox id="Amount" ReadOnly = "true" Runat="server"/>

<asp:Button id="btnCalculate" Text="Calculate" OnClick="Calculate" Runat="server"/>

</form>

</body>

</html>

Events_cb.vb

Imports System

Imports System.Collections

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class BaseClass

Inherits System.Web.UI.Page

'Each control used on events.aspx to be declared here with same id

Protected qty as textbox

Protected price as textbox

Protected amount as textbox

Sub Page_Load(Source As Object, E As EventArgs)

'this is the page load event

'gets fired each time the page is loaded

response.write("Page Load Event : -->fired
")

if NOT (isPostBack)

'the code here gets fired only one at page load

'subsequent reloads do not fire it due to the not isPostBack construct

response.write("The not isPostBack construct:--> ensures this does not get fired at reloads")

end if

End Sub

'This sub moved from the events.aspx form

Sub Calculate(src As Object, e As EventArgs)

Amount.Text = Cstr(cint(qty.text)*cint(price.text))

End Sub

End Class

Let's discuss the example in detail:

- 1. I have defined a class called BaseClass in the Code Behind file and moved the Calculate sub from the aspx form into this class. This class inherits from the System.Web.UI.Page.
- 2. I will be extracting the text value of textboxes Qty and Price, multi-plying the two, and putting the result in the Amount textbox. Since I need to access the property values of these three textboxes from my Code Behind file, I declare three textboxes with the same id in the Code Behind file like this:
 - 3. Protected qty as textbox
 - 4. Protected price as textbox

Protected amount as textbox

5. The Qty, Price, and Amount textboxes are WebControls since I have initialized them with the asp: tag prefix. For example, the Qty textbox is created as follows:

<asp:TextBox id="Qty" Runat="server"/>

Controls exist in the System.Web.UI.WebControls namespace hence I must import this namespace before I can access their properties by code. This is done by the import directive at the top of the page:

Imports System.Web.UI.WebControls

 Finally, I have coded the Page_Load Event to display a message when it gets fired. This event gets fired each time the page gets loaded. At times we need to code events that get fired only at the initial page load and not on subsequent reloads. For example, we can bind a Web Control to a

data source (I will be discussing data binding in <u>Chapter 4</u>) and want the binding to occur only once at page load.

The isPostBack property of the page lets us determine if posting has already occurred to the page. Thus we use the following construct to display a message only on the first load of the page:

IF NOT (isPostBack)

response.write("The not isPostBack construct:-->.....")

End If

Server Controls

There have been many attempts to encapsulate HTML rendering into controls. We have had objects like VBXs, OLE controls, and ActiveX controls, all of which attempted to give us a simple way to generate HTML. The problem with these controls is that they made the presumption that the users accessing our sites would have the very latest browsers. The server side controls introduced with ASP.NET make no such requirement of the browser. They render pure HTML to the browser, thus overcoming the shortcoming of its client side brethren. These server controls are fully encapsulated objects that expose events, properties, and methods to programmatic access. They exist independent of the web form on which they are drawn.

ASP.NET provides two sets of controls: HTML and Web Controls. HTML controls correspond to traditional HTML controls with the same name. Web Controls provide features such as automatic browser detection, a consistent object model, and data binding capabilities.

HTML Controls

HTML controls belong to the System.Web.UI.HTMLControls namespace and derive from the HTMLControl base class. They are initiated with the runat = "server" attribute. For example, the following HTML creates an instance of a HTMLInputText named textbox 1.

<Input type = "text" runat="server" id = "textbox1" value = "test">

These controls map directly to their HTML counterparts and are useful for providing backward compatibility with ASP. These controls do not provide any abstraction like their Web Control counterpart and do not automatically detect the requesting browser to modify the HTML they render. The main use of these controls is to provide a quick migration path to ASP.NET as existing HTML tags can be upgraded to server controls just by supplying the runat = "server" attribute.

I have provided examples of various HTML controls in the following example with a detailed discussion of each one afterwards. <u>Figure 2.2</u> shows various HTML controls.



<%@ Page Language="vb" %>

<html>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<H2>HTML Controls</H2>

<form method="POST" runat="server">

HTMLAnchor Control

>

 HTMLButton

</button>

>

HTMLImage

HTMLInputButton:

<input type=submit value="Submit" runat=server/>

>HTMLInputCheckBox:

<input id="Check1" type=checkbox runat="server"/>

>

HTMLInputHidden :(hidden)

<input id="HiddenValue" type=hidden value="Initial Value" runat=server>

>

HtmlInputImage:

<input type=image id="InputImage1" src="/quickstart/aspplus/images/ mango.jpg" runat="server"/>

>
:HtmlInputRadioButton:

<input type="radio" id="Radio1" name="Mode" runat="server"/>Option 1

<input type="radio" id="Radio2" name="Mode" runat="server"/>Option 2

HtmlInputText (password)

<input id="Password" type=password size=40 runat=server>

>tr>
HtmlInputText

<input id="Name" type=text size=40 runat=server>

>HTMLSelect :

<select id="Name_sel" runat="server">

<option>Hersh</option>

<option>Ritu</option>

<option>Ria</option>

<option>Bhasin</option>

</select>

</form>

</body>

</html>

1. HTMLForm

Corresponding HTML tag: <form> Example: <form method="POST" runat="server"> 2. HTMLAnchor Corresponding HTML tag: <a>

Example:

HTMLAnchor Control

3. HTMLButton

Corresponding HTML tag: <button>

Example:

<button id="Button1" style="font: 8pt verdana;backgroundcolor:lightgreen;border-color:black;height=30;width:100" runat="server">

 HTMLButton </button> 4. **HTMLImage**

Corresponding HTML tag:

Example:

```
<img ID="Image1" src="/quickstart/aspplus/images/cereal1.gif"
runat="server"/>
5. HTMLInputButton
```

Corresponding HTML tag: <input type = "button">

Example:

```
<input type=submit value="Submit" runat=server/>
6. HTMLInputCheckBox
Corresponding HTML tag: <input type = "check">
Example:
```

<input id="Check1" type=checkbox runat="server"/>
7. HTMLInputHidden
Corresponding HTML tag: <input type = "hidden">

Example:

```
<input id="HiddenValue" type=hidden value="Initial Value" runat=server>
    8. HTMLInputImage
Corresponding HTML tag: <input type = "image">
```

Example:

```
<input type=image id="InputImage1"
src="/quickstart/aspplus/images/mango.jpg" runat="server"/>
9. HTMLInputRadioButton
```

Corresponding HTML tag: <input type = "radio">

Example:

```
<input type="radio" id="Radio1" name="Mode" runat="server"/>Option 1<br>
```

<input type="radio" id="Radio2" name="Mode" runat="server"/>Option 2

10. HTMLInputText (password)

Corresponding HTML tag: <input type = "password">

Example:

```
<input id="Password" type=password size=40 runat=server>
11. HTMLInputText
```

Corresponding HTML tag: <input type = "text">

Example:

```
<input id="Name" type=text size=40 runat=server>
12. HTMLSelect
Corresponding HTML tag: <select>
```

Example:

<select id="Name_sel" runat="server">

- <option>Hersh</option>
- <option>Ritu</option>
- <option>Ria</option>
- <option>Bhasin</option>

</select>

Web Controls

A Web Control is created by adding a prefix of asp before a control. This prefix is actually a namespace of the run time control. The remainder of the tag is the name of the run time control itself. Like the HTML controls these controls also contain a runat = "server" attribute. This is an example of a textbox Web Control:

<asp:TextBox runat="server" id = textbox1 value ="test"/>

Since HTML controls can be used server side, we might question the need for another set of controls that provide similar functionality. The idea with Web Controls is that it simplifies the nomenclature of a control. Controls that overlapped in their functionality have been reduced to a single control. For example, consider the three input controls in the following:

<input id="Name" type=text size=40 runat=server>

<input id="Password" type=password size=40 runat=server>

<textarea id="TextArea1" cols=40 rows=4 runat=server />

Each of these controls are used for accepting input from the user, however, there is no consistency. Web Controls provide a more logical solution.

<asp:TextBox id="Text1" runat="server"/>

<asp:TextBox id="Text2" TextMode="Password" runat="server"/>

<asp:TextBox id="Text3" TextMode="Multiline" rows="4" runat="server"/> Now one control provides the functionality of three. This new syntax is much easier to remember. Furthermore, the WebControl base class from which the Web Controls derive implement functionality that is common to all Web Controls. Properties such as ForeColor, Font, BackColor, Selected, Enabled, etc., are consistent among controls. The developer thus needs to remember one property setting that he can apply to all controls. These controls provide automatic browser detection. They can customize their capabilities to match the calling browser requirements. As we will see in <u>Chapter 4</u>, Web Controls like the DataGrid and DataList can be bound to a data source and can make HTML rendering a breeze.

Web Controls are of the following types:

- Intrinsic controls The rationalized HTML controls like Text Boxes, DropDownLists, etc., that start with asp: prefix.
- ListBound controls Controls that simplify the task of generating Html for a repeating data source. These are controls like the DataGrid, DataList, Repeater and DataReader.
- <u>Rich controls</u> These consist of the Calendar and AdRotator. The Calendar outputs HTML for downlevel browsers or DHTML for uplevel browsers. The AdRotator displays rotating advertisements.
- <u>Validation controls</u> These include controls such as Compare Validator, Range Validator, RequiredField Validator, RegularExpression Validator, Custom Validator and Validation Summary. These controls aim to simplify the process of user input validation.

I will be discussing Intrinsic and Rich controls here. ListBound controls are discussed in <u>Chapter 3</u>, "<u>Using ADO.NET in the .NET Framework.</u>" and validation controls are the subject matter of <u>Chapter 5</u>, "<u>Input Validation.</u>"

Intrinsic Controls

There are a number of intrinsic Web Controls supplied with ASP.NET. These include controls like TextBoxes, CheckBoxes, Radio Buttons, and DropDown Lists to name a few. <u>Figure 2.3</u> shows some of the intrinsic controls. I discuss these controls in detail in this section.

| http://locallout/Asphellicsk/Chapter2/samples2/Web_jet | vensic_aspe - Microsoft Informat Explorer | .0 |
|---|--|-------|
| 27 2 2 Sant - mentagine | nnesttis Yanovi | 82 |
| Adamss 2 traps/hosher/Aspletbook/Chapter/Samples/Meth.uk | NK.K01 - | 00 |
| Links @]fop-Computers-Programming @]CustomerLinks @]P | ee humal @]Search Engine Submission States @]Windows Media @]W | nters |
| Intrensic Web Controls | | Î |
| | | |
| Text Box | MultiLine: | |
| Check Box
IF CheckBox1
Rodie Betton
F Rediebutton1 IC Rediobutton2 | | |
| CheckBexLists
P Choice1 | | 1 |
| Choice2
RedieduttonList
* Choice1 C Choice2 | | |
| Choice1
Choice2 | | |
|] Cone | () Local infrared | |

Figure 2.3: Web Controls.

The script for creating these controls can be found in the Web_intrinsic.aspx file, the listing of which follows.

<%@ Import Namespace="System.Data" %>

<html>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<h3>Intrinsic Web Controls</h3>

<form runat="server">

Text Box:<asp:TextBox id="Text1" runat="server"/>

Password :<asp:TextBox id="Text2" TextMode="Password" runat="server"/>

MultiLine: <asp:TextBox id="Text3" TextMode="Multiline" rows="4" runat="server"/>

<hr>

Check Box

<asp:CheckBox runat="server" Text="CheckBox1" Checked="True"></asp:CheckBox>

Radio Button

<asp:RadioButton runat="server" Text="RadioButton1" GroupName="Group1" Checked="true">

</asp:RadioButton>

<asp:RadioButton runat="server" Text="RadioButton2" GroupName="Group1"></asp:RadioButton>

<hr>

CheckBoxLists

<asp:CheckBoxList runat="server">

<asp:ListItem Text="Choice1" Value="1" selected="true"/>

<asp:ListItem Text="Choice2" Value="2" selected="true"/>

</asp:CheckBoxList >

RadioButtonList

<asp:RadioButtonList runat="server" RepeatDirection= "horizontal">

<asp:ListItem Text="Choice1" Value="1" selected="true"/>

<asp:ListItem Text="Choice2" Value="2"/>

</asp:RadioButtonList >

<hr>

DropDownList

<asp:DropDownList runat="server">

<asp:ListItem Text="Choice1" Value="1" selected="true"/>

<asp:ListItem Text="Choice2" Value="2"/>

</asp:DropDownList>

ListBox

<asp:ListBox runat="server" SelectionMode="Multiple">

<asp:ListItem Text="Choice1" Value="1" selected="true"/>

<asp:ListItem Text="Choice2" Value="2"/>

</asp:ListBox>

<hr>

Button :<asp:Button runat="server" Text="Click Me"></asp:Button>

LinkButton :<asp:LinkButton runat="server" Text="Click Me"></asp:linkButton>

ImageButton: <asp:ImageButton runat="server" ImageUrl="gifcon.gif"></asp:ImageButton>

Hyperlink : <asp:HyperLink runat="server" Text="HyperLink"

NavigateUrl="web_intrensic.aspx"></asp:HyperLink>

Image :<asp:Image runat="server" ImageUrl="gifcon.gif"></asp:Image>

<asp:Panel runat="server"></asp:Panel>

<hr>

Table

<asp:Table runat="server" GridLines="Both" BorderWidth="1px">

<asp:TableRow>

<asp:TableCell>Column1</asp:TableCell>

<asp:TableCell>Column2</asp:TableCell>

</asp:TableRow>

<asp:TableRow>

<asp:TableCell>Column3</asp:TableCell>

<asp:TableCell>Column4</asp:TableCell>

</asp:TableRow>

</asp:Table>

</form>

</body>

</html>

I now discuss the various controls covered under this category.

Text Boxes

Textboxes accept input from users. They can be bound to data. A text box's appearance can be controlled by its TextMode property. The TextMode property can either be single-line, multi-line, or password. When it is set to multi-line, the row's property controls how many lines of input can be accepted from the user. Here are a few examples:

Text Box: <asp:TextBox id="Text1" runat="server"/>

Password: <asp:TextBox id="Text2" TextMode="Password" runat="server"/>

A check box has the following properties:

- Checked: can be either true or false. When true, it is in the checked state.
- Text: This is the text that gets displayed beside the check box.

- AutoPostBack: If this property is true, it causes an immediate postback to the server when the check box's checked property is changed.
- TextAlign: Sets the alignment of the check box. Can be left or right.
- Example:

<asp:CheckBox runat="server" Text="CheckBox1" Checked = "True">

</asp:CheckBox>

Radio Button

A radio button is similar to a check box. However, you can display a number of radio buttons, relating them to the same group. The user will then only be able to select one radio button from the group.

Example:

Radio Button

<asp:RadioButton runat="server" Text="RadioButton1" GroupName = "Group1" Checked="true">

</asp:RadioButton>

<asp:RadioButton runat="server" Text="RadioButton2" GroupName="Group1">

</asp:RadioButton>

CheckBoxList

This control provides a multi-selection checked list. It has the following properties:

- Selected Property: This property can be checked to determine the selected item.
- RepeatDirection: Can be horizontal or vertical. Specifies direction of control rendering.
- RepeatLayout: Can be Table or Flow. Table means the list will be rendered within a table. Flow renders it without a table structure.

Example:

<asp:CheckBoxList runat="server">

<asp:ListItem Text="Choice1" Value="1" selected="true"/> <asp:ListItem Text="Choice2" Value="2" selected="true"/> </asp:CheckBoxList >

RadioButtonList

This control provides a group of radio buttons, which allows only one selected value. It has the following properties:

- Selected Property: This property can be checked to determine the selected item.
- RepeatDirection: Can be horizontal or vertical. Specifies direction of control rendering.
- RepeatLayout: Can be Table or Flow. Table means that list will be rendered within a table. Flow renders it without a table structure.

Example:

<asp:RadioButtonList runat="server" RepeatDirection= "horizontal"> <asp:ListItem Text="Choice1" Value="1" selected="true"/> <asp:ListItem Text="Choice2" Value="2"/> </asp:RadioButtonList >

DropDownList

The DropDownList displays only one item in a list at a time. It can be bound to a data source.

Examples:

o Adding to the list using ListItem

<asp:DropDownList id= "dropdown" runat="server">

<asp:ListItem Text="Choice1" Value="1" selected="true"/>

<asp:ListItem Text="Choice2" Value="2"/>

</asp:DropDownList>

- o Adding to the list using code
- Sub AddtoList()
- DropDown.Items.Add("choice1")
- DropDown.Items.Add("choice2")
- DropDown.Items.Add("choice3")

End Sub

- o Binding to a DataSource
- Sub Page_Load(sender As Object, e As EventArgs)
- o If Not IsPostBack Then
- Dim values as ArrayList= new ArrayList()
- values.Add ("Choice1")
- values.Add ("Choice2")
- values.Add ("Choice3")
- DropDown.DataSource = values
- o DropDown.DataBind
- o End If

End Sub

Extracting the Selected Value

textbox1.text = DropDown.SelectedItem.Text • List Box

The List Box control renders a scrollable list of values. It allows either a single option or multiple options to be selected. Multiple selection is enabled by setting the SelectionMode property to "Multiple". Examples:

- Adding to the list using ListItem
 - <asp:ListBox id="list1" runat="server" SelectionMode="Multiple">
 - <asp:ListItem Text="Choice1" Value="1" selected="true"/>
 - <asp:ListItem Text="Choice2" Value="2"/>

</asp:ListBox>

- o Adding to the list using code
- Sub AddtoList()
- List1.Items.Add("choice1")
- List1.Items.Add("choice2")
- List1.Items.Add("choice3")

| End \$ | Sub |
|--------|-----|
|--------|-----|

| o Binding to | a DataSource |
|--------------|--------------|
|--------------|--------------|

- Sub Page_Load(sender As Object, e As EventArgs)
- o If Not IsPostBack Then
- Dim values as ArrayList= new ArrayList()
- values.Add ("Choice1")
- values.Add ("Choice2")
- values.Add ("Choice3")
- List1.DataSource = values
- o List1.DataBind
- o End If
- End Sub
- o Extracting the Selected Value
- textbox1.text = List1.SelectedItem.Text
 - Extracting Multiple values (when multi-selection is enabled)
 - Sub GetMultiples()
 - o Dim item As ListItem
 - Dim s As String = ""
 - For Each item In List1.Items
 - o If item.Selected Then
 - o s += item.Text + "
>"
 - o End If
 - o Next
 - Response.write(s)

End Sub

Button

Generates a standard 3D Push button used to submit the page back to the server.

Example:

<asp:Button runat="server" Text="Click Me">

</asp:Button>

LinkButton

The LinkButton has a similar functionality as that of a Button control. It is also used to submit a page back to the server. However, instead of a button, a link is generated.

Example:

<asp:LinkButton runat="server" Text="Click Me">

</asp:linkButton>

ImageButton

The ImageButton renders a clickable image, which posts back to the server. Use this control when you want to display a picture instead of a button. Example:

<asp:ImageButton runat="server" ImageUrl="gifcon.gif">

</asp:ImageButton>

HyperLink

This control displays a hyperlink, which allows the user to navigate to other URLs.

Example:

<asp:HyperLink runat="server" Text="HyperLink"

NavigateUrl="web_intrensic.aspx">

</asp:HyperLink>

Image

The Image control is used to display an image in the page. Example:

<asp:Image runat="server" ImageUrl="gifcon.gif"></asp:Image>

Table

The Table control, along with TableRow and TableCell controls,

programmatically renders tables.

Example:

Table

<asp:Table runat="server" GridLines="Both" BorderWidth="1px"> <asp:TableRow>

<asp:TableCell>Column1</asp:TableCell>

<asp:TableCell>Column2</asp:TableCell>

</asp:TableRow>

<asp:TableRow>

<asp:TableCell>Column3</asp:TableCell>

<asp:TableCell>Column4</asp:TableCell>

</asp:TableRow>

</asp:Table>

Panel

The Panel is used as a container for other controls. A number of controls can be put within a panel tag and made visible or invisible simply by setting the Visible property to true or false. In Figure 2.4 an example of a panel is given.

| (a) http://ksalkest/AspNelliosk/Chapter2/samples2/panel.espx - Microsoft Internet Explorer | |
|---|--------------------|
| Search + | na * 62 |
| Charl . O C & Qiearth @Favortes @Hetary C-@ E | |
| Addmss 🕼 http://kicahiid.aqhiet8niid/higher2/samples2/panel.agor | • බංග |
| Links @]fup-Computers-Programming @]CustomerLinks @]PreeHonnel @]Sourch Engine Subrassion Stes @]Window | s Media ∉] Windows |
| Panci
Cty: Proc. Anount: Show Parel Hide Parel | |
| a) Dose | acalistranet |





<%@ Page Language="vb" %>

<html>

<head>

<script Runat="server">

Sub show(src As Object, e As EventArgs)

detailsPanel.visible = "true"

End sub

Sub Hide(src As Object, e As EventArgs)

detailsPanel.visible = "false"

End sub

</script>

</head>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<H2> Panel</H2>

<form method="POST" runat="server">

<asp:Panel id="detailsPanel" runat="server" Visible="false">

Qty:<asp:TextBox id="Qty" Runat="server"/>

Price: <asp:TextBox id="Price" Runat="server"/>

Amount:<asp:TextBox id="Amount" ReadOnly = "true" Runat="server"/>

</asp:panel>

<asp:Button id="btnShow" Text="Show Panel" OnClick="Show" Runat="server"/>

<asp:Button id="btnhide" Text="Hide Panel" OnClick="Hide" Runat="server"/>

</form>

</body>

</html>

In this example, the textboxes on the form are put within a Panel tag. The "Show" function set the visible property of the panel to true. This has the effect of showing all

the textboxes on the form. Likewise the "Hide" function sets the visible property of the panel to false and hides all the textboxes.

Rich Controls

There are two controls known as "Rich Controls" supplied with ASP.NET. It is expected that more controls will soon be forthcoming. The AdRotator displays a sequence of advertisements. The Calendar Control makes it easy to provide date and date navigation functionality.

AdRotator

The AdRotator is a control that produces banner advertisements. Clicking on the control navigates the browser to a specified URL. A different advertisement is loaded each time the page is loaded in the browser. You need to set up an XML file like the following:

Ads.XML

<Advertisements>

<Ad>

<ImageUrl>sos.gif</ImageUrl>

- <TargetUrl>www.someurl.com</TargetUrl>
- <AlternateText>Sos</AlternateText>
- <Keyword>sos</Keyword>

<Impressions>80</Impressions>

</Ad>

<Ad>

```
<ImageUrl>crylogo.gif</ImageUrl>
<TargetUrl>www.someurl.com</TargetUrl>
<AlternateText>crylogo</AlternateText>
<Keyword>crylogo</Keyword>
<Impressions>80</Impressions>
</Ad>
</Advertisements>
```

The following properties are set in the XML file. Only the ImageUrl attribute is required; others are optional.

ImageUrl: The url pointing to the advertisement to be displayed.

TargetUrl: The url to navigate to (when the advertisement is clicked).

AlternateText: The "ALT" attribute.

Impressions: The weight relative to the image to be displayed. The greater the weight, the more times the ad will be displayed. In Figure 2.5, you can see what the adRotator looks like.



Figure 2.5: AdRotator.

You initiate the adRotator component as follows: adrotator.aspx

<html>

```
<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">
```

```
<h3><font face="Verdana">AdRotator Control</font></h3>
```

<form runat=server>

```
<asp:AdRotator id="ar1" AdvertisementFile="Ads.xml" BorderWidth="1" runat=server />
```

</form>

</body>

</html>

Calendar Control

The Calendar control allows the user to select a certain date, all days in a week or month. It is very easy to set up; as simple as saying:

```
<asp:Calendar runat = "server"/>
```

This gives a no frills basic calendar. However, as in other controls, much of its functionality can be customized. Here is an example.

Calendar.aspx

<%@ Page Language="VB" %> <html> <head> <script runat="server"> Sub myCal_OnSelectionChanged(Source As Object, E As EventArgs) Dim sel As Integer Dim i as integer Dim s as string sel = Mycal.SelectedDates.Count

```
s = ""
    For i = 0 to sel-1
      'Beta 2 Code lowercase "d" gives shortdate
       'Uppercase "D" will give long date
       s = s + Mycal.SelectedDates.Item(i).ToString("d")
    next
   selDate.Text = s
  End Sub
  Sub mycal_OnVisibleMonthChanged(Source As Object, E As
MonthChangedEventArgs)
     selDate.Text = "Current Month " + " " + E.NewDate.ToString() _
    + " Previous Month : " + E.PreviousDate.ToString()
   End Sub
  </script>
 </head>
 <body style="background-color='beige'; font-family='verdana'; font-
size='10pt'">
  <form id="myForm" runat="server">
   <asp:Label id="selDate" runat="server" />
  <ASP:Calendar id="myCal" runat="server"
      BackColor="#666699" ForeColor="black"
      BorderWidth="2"
      BorderStyle="Solid" BorderColor="Black"
      CellSpacing=2 CellPadding=2
      ShowGridLines=true
      TitleStyle-BorderColor="#6666699"
      TitleStyle-BorderWidth="3"
      TitleStyle-BackColor="DarkGray"
      TitleStyle-Height="50px"
      DayHeaderStyle-BorderColor="teal"
      DavHeaderStvle-BorderWidth="3"
      DayHeaderStyle-BackColor="teal"
       DayHeaderStyle-ForeColor="black"
      DayHeaderStyle-Height="20px"
      DayStyle-Width="50px"
      DayStyle-Height="50px"
      TodayDayStyle-BorderWidth="3"
      WeekEndDayStyle-BackColor="teal"
      WeekEndDayStyle-Width="50px"
      WeekEndDayStyle-Height="50px"
      SelectedDayStyle-BorderColor="firebrick"
      SelectedDayStyle-BorderWidth="3"
      OtherMonthDayStyle-Width="50px"
      OtherMonthDayStyle-Height="50px"
      OnSelectionChanged="myCal_OnSelectionChanged"
      OnVisibleMonthChanged="mycal_OnVisibleMonthChanged"
      runat="server"
      SelectionMode = "DayWeekMonth"/>
  </form>
```



None: No date selection allowed.

Day:Only a certain day can be selected.

DayWeek: A day or a complete week can be selected.

DayWeekMonth: A day, complete week, or whole month, can be selected. By default this property is set to "Day."

	o+Coepco	ers - Program	uning el	Oustionize Uit	As ∉]Mo	e Hotiwal	@]South (ngire Subraskin Stas 🖉 (Mindows Meda 🦿)	rindovs
8			Janua	ry 2001			≥		
2	Sin	Mon	Tue	Wed	Thu	Fri	Sat		
2	31	1	2	3	4	2	é		
N.	Z	8	2	10	ш	12	13		
2	14	15	<u>16</u>	17	18	12	20		
2	21	22	23	24	25	26	27		
2	28	29	30	31	1	2	3		
2	9	2	2	Z	8	2	10		

Figure 2.6: Calendar.

OnSelectionChanged Event:

This event is fired each time the user makes a day, month, or week selection. I have specified myCal_OnSelectionChanged sub to fire on this event. The SelectedDates.Count method extracts the number of days selected. It can make a single day (SelectionMode = "Day"), a complete Week (SelectionMode = "DayWeek"), or a complete month (SelectionMode =

"DayWeekMonth"). I then look through the days selected and build a string which extracts the days

and displays it in a TextBox like the following:

For i = 0 to sel-1

s = s + Mycal.SelectedDates.Item(i).ToString("d") next

selDate.Text = s

OnVisibleMonthChanged Event:

This event gets fired each time a month is changed (say you switch from January to February). I have specified the ${\tt sub}$

cal_OnVisibleMonthChanged sub to fire on this event. This sub just

displays the current month and the previous month. The code for this follows:

Sub mycal_OnVisibleMonthChanged(Source As Object, E As MonthChangedEventArgs)

selDate.Text = "Current Month " + " " + E.NewDate.ToString() _

" Previous Month : " + E.PreviousDate.ToString() End Sub

Summary

ASP.NET controls encourage good programming habits. You can logically separate presentation from code thus avoiding "spaghetti" code. These controls are consistent in their nomenclature. We have to remember a core set of functionality that is applicable to all controls. These controls can target any browser, handheld device, or cell phone as they have the server generate the user interface and send out pure HTML to the browser. Since every browser understands HTML we can write applications that can target all client sites irrespective of their browser preferences.

Chapter 3: Using ADO.NET in the .NET Framework

Overview

ADO.NET is a technique whose time has come. If you reflect on ADO, and how it evolved over time, you will realize that it was a child of the client-server era. It was steeped in the connection-based method of handling data. A typical database access operation would open a database connection and leave it open until the looping operation of the recordset was completed. This method was not suitable for Web programming which required a disconnected way of manipulating data. This was because a connection-based data access methodology required a connection to be kept alive for each client connection. Multiple clients demanding resources from a server would very soon bring it to its knees. This led to the development of RDS (Remote Data Services). With the advent of XML, the request/response paradigm became the order of the day. To keep up with this message-based system of communication HTTP support was added to RDS, which allowed business logic to be called at the middle tier. XML data and database data however follow different data models. XML follows a heterogeneous and hierarchical data model (XMLDOM) whereas MDAC (Microsoft Data Access Technologies) follows a relational model. Developers were now faced with the added complexity of choosing between MSXML and MDAC to work with XML data.

Bound controls were introduced in Visual InterDev. These controls were used to connect to a datasource with limited programming effort. For example, you could drag and drop a Bound control onto an ASP page and bind it to a datasource (by making a visual property selection) in a Visual Basic style. This had one major drawback. Visual InterDev generated a lot of script behind the scenes. This would turn away many of us from ever using this technique.

These piecemeal changes being made to ADO were pushing it to its limits. The time had come to rewrite this technology from the ground up and the result is ADO.NET.

ADO.NET has XML support built in at a very basic level and working with XML using this technology is quite similar to working with database data. No longer is choosing between MDAC and MSXML an issue. ADO.NET uses a disconnected way of working with database data and avoids the performance penalty associated with ADO. It has revitalized this concept of using bound controls and understands what it has to do in relation to a Bound control without maintaining tons of script in supplementary files. In fact it is so lean that you can use all the Bound controls and still use a basic editor like Notepad for your development efforts. ADO is still available (if you should want to use it) through the .NET COM interoperability services. However, programming in ADO.NET is quite similar to ADO, thus developers will not be faced with a steep learning curve. There are four basic things that we need to do with data. Query it, add to it, update it, and delete from it. In the process we need controls to present data and make it available to the user. In this chapter, I will discuss various facets of ADO.NET that allow us to accomplish these requirements. I will commence with discussing the two protagonists of the ADO.NET architecture; DataSets (the disconnected layer) and Managed Providers (the connected layer).

The DataSet

The recordset has been retired. A new object has been introduced in ADO.NET, which is called the DataSet. You can think of the *DataSet* as an in-memory copy of the database, complete with tables, columns, relationships, constraints, and data. It allows for relationships to exist between multiple tables, analogous to a foreign-key relationship in a database. You can navigate between tables based upon their relationships. In ADO one had to rely on SQL commands like Join to relate multiple tables and navigation involved jumping sequentially from record to record.

A DataSet is not connected to any database and has no knowledge of the source of its data. An ADO recordset is connected to the database through an OLE DB provider. In ADO.NET, you communicate with the database through Command objects, the code of which can be modified. A DataSet can talk to a variety of datasources. It can receive data from a database, an XML file, from code, or user input. No matter what the source of the data within the DataSet is, it is manipulated through the same set of standard APIs.

Since DataSets are transmitted as an XML stream, DataSets can be passed between any objects (not just COM objects) without interference from firewalls. To transmit an ADO disconnected recordset from one component to another, COM marshalling had to be used (marshalling is the process of gathering data from one more applications, storing the data pieces in a memory buffer and converting it to the format that a receiving application would understand).

A DataSet contains a table collection (the DataTables collection), which in turn contains a column collection. A DataSet also contains a row collection, which contains all rows retrieved from the datasource.

Managed Providers

A DataSet is "blind" as to where the data comes from or where it may go. It is the responsibility of the Managed Providers to have knowledge to enable the DataSet to interact with the datasource (which can be a database, an XML file, or user input). A *Managed Provider* consists of Connection, Command, DataReader, and DataAdapter classes. The connection and command objects are similar to their ADO namesakes. The DataReader is like a forward, read-only recordset and the DataAdapter is the bridge between the Managed Provider and the DataSet. I shall be dealing with each of these as I go along.

There are two Managed Providers in ADO.NET:

- A provider optimized for SQL Server 7 or higher.
- An OLE DB provider for accessing data other than SQL Server 7 or higher (though you can also use it for SQL Server).

The SQL Server provider provides direct access to a SQL Server using a protocol called TDS (*Tabular Data System*). This provides enhanced benefits when using SQL server. The OLE DB provider is more generic, and though it can be used with a SQL Server database, it will not provide any performance enhancements.

In this chapter, I have mostly used the OLE DB provider. Using the SQL provider is quite similar. In most cases, all you need to do is replace all occurrences of the word "OleDb" with "Sql" and import the namespace System.Data.SqlClient instead of System.Data.OleDb.

There are five basic steps involved in building a web form to interact with the data:

- 1. Import the relevant Namespaces.
- 2. Connect to the database using the Connection Object (SqlConnection or OleDbConnection).
- 3. Populate the DataAdapter Object (SqlDataAdapter or OleDbDataAdapter) with data from the datasource.

- 4. Populate a DataSet using the Fill method of DataAdapter.
- 5. Bind the server control like DataGrid to the DataSet.

I will build a web form to demonstrate these steps. This form will retrieve data from the master's table and display the result in a grid format. <u>Figure 3.1</u> shows what the outcome will look like.

	Catorize Lriss 20	Free Hoteral 2	in arch	Engine	interiosion Site	(U)W	Indones Med	ta වුහා	ndows		
Maste	es Table										1
code_	value code_duplay	code_categor	7 37	e clore	gopening						
18	Wells Fargo	605	A	10	0						
19	Selary	702	I	20	0						
2	Utiltics	704	Е	10	0						
6	Vsta	605	А	0	0						
7	Discover Car	d/605	A	0	0						

Figure 3.1: Interacting with Data. MastersGrid.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<head>

<H4>Masters Table</H4>

<script language="VB" runat="server">

Sub Page_Load(Source As Object, E As EventArgs)

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

'Connection syntax

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET; User ID=sa"

myConnection = New OleDbConnection(ConnStr)

'DataSetCommand

SQL = "select * from Masters"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "Masters")

'Binding a Grid

DataGrid1.DataSource=ds.Tables("Masters").DefaultView

DataGrid1.DataBind()

End Sub

</script>

</head>

<body>

<form runat=server>

<asp:DataGrid id="DataGrid1" runat="server" />

</form>

</body>

</html>

I will now discuss the script and introduce you to various steps involved in using ADO.NET to interact with the database.

Namespace

Namespaces can be thought of as using references in Visual Basic. You need to put a construct at the top of your page.

For the OLE DB provider, use the following: <%@ Import Namespace="System.Data" %> <%@ Import Namespace="System.Data.OleDb" %> For the SQL provider, use the following: <%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.SqlClient" %>

The Connection Object

A connection is opened implicitly (as in the preceding example) when using a DataAdapter or explicitly by calling the Open method on the connection as in the following example:

OpenExplicit.aspx

<%@ Import Namespace="System.Data" %>

```
<%@ Import Namespace="System.Data.OleDb" %>
```

<html>

<script language="VB" runat="server">

Dim myConnection As OleDbConnection

Dim SQL As String

Dim ConnStr As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

try

Dim mycommand As New OleDbCommand(sql,myConnection)

myConnection.Open()

Response.Write("Opened Connection to " + MyConnection.ConnectionString + "

catch myException as Exception

Response.Write("Exception: " + myException.ToString())

finally

'Close the connection explicitly

Response.Write("Closed Connection. It is important to close connections explicitly.")

myConnection.Close()

end try

End Sub

</script>

</html>
The DataAdapter Object

A DataSet has no knowledge of the datasource. It keeps track of the changes made to it and systematically exposes it to the DataAdapter, which can then apply these changes to the datasource in an optimistic manner.

Populating the DataAdapter object in ADO.NET is similar to populating a command object in ADO. You pass the DataAdapter object a SQL query string and a connection object such as in the following example:

SQL = "select * from Masters"

myCommand = New OleDbDataAdapter(SQL, myConnection)

You can see that the syntax for the DataAdapter looks exactly like the command object syntax in ADO. The appropriate provider's DataAdapter object is populated with the result of the SQL query statement. An ADO command can only accept one command at a time, whereas the DataAdapter can be assigned update, delete, and insert commands. These properties are used when the DataAdapter update method is called. An update/insert/delete in the DataSet calls the appropriate update/insert/delete command of the DataAdapter. We can associate a set of stored procedures to perform an update, insert, or delete which will fire whenever the update method is called on the DataAdapter.

The Fill method acts as the bridge between the datasource and the DataSet. It loads the data stored in the DataAdapter into the DataSet. It requires two parameters: the DataSet name, and the name of the table (in the DataSet) into which to load the data.

The syntax follows:

myCommand.Fill(ds, "Masters")

Finally the DataGrid is bound to the DataSet.

DataGrid1.DataSource=ds.Tables("Masters").DefaultView

DataGrid1.DataBind()

The Command Object

A command is a SQL query, a stored procedure, or a table name that is issued to the database. The execution of the command can return results from the database and this resultset can then be passed onto another object such as the DataReader or the DataAdapter. This in turn can act as a datasource for a Bound control like the DataGrid. The command can also be an "action" query like an insert, update, or delete query that does not return any results. The command object for use with MS SQL Server is called SqlCommand and for use with OLE DB providers is called OleDbCommand.

The command object is constructed by providing a SQL query and a connection object as parameters to the command object. This is shown in the following example:

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET; User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

sql = "delete from masters where code_display = 'test'"

Dim mycommand As New OleDbCommand(sql,myConnection)

myConnection.Open()

myCommand.ExecuteNonQuery()

myConnection.Close()

CommandType

The command object has a property called CommandType that is used to define the type of command being sent to the database. There are three CommandTypes available. These are the Text, StoredProcedure, and TableDirect types.

The Text CommandType is the default. This is a string of SQL text that is issued to the database as in the following example (I did not explicitly specify the CommandType in this example, as it is the default):

sql = "delete from masters where code_display = 'test'"

Dim mycommand As New OleDbCommand(sql,myConnection) The StoredProcedure CommandType is used to execute a database-stored procedure. You can pass parameters to the stored procedure using the parameters collection of the command object (this will be discussed in detail later in this chapter) as in the following example:

Dim myCommand As New OleDbCommand("p_authors", myconnection)

myCommand.CommandType = CommandType.StoredProcedure

objParam = myCommand.Parameters.Add("State", OleDbType.VarChar, 10)

objParam.Direction = ParameterDirection.Input

objParam.Value = "CA"

The TableDirect CommandType is used to provide a table name to the command object as in the following example:

Dim myCommand As New OleDbCommand("Groups", myconnection)

myCommand.CommandType = CommandType.TableDirect

Executing Commands

The command object provides methods to execute the SQL statement, stored procedure or return the records from a table name provided to it by the CommandType property. There are three methods available. These are <u>ExecuteNonQuery</u>, <u>ExecuteReader</u>, and <u>ExecuteScalar</u>. In addition the SqlCommand class provides two additional methods. These are the ExecuteResultSet and ExecuteXmlReader. The ExecuteResultSet is reserved for future use and is consequently not available for use.

ExecuteNonQuery

This method is used when a result set is not to be returned from the database, for example:

Dim mycommand As New OleDbCommand(_

"UPDATE Masters Set Opening = 90 WHERE code_display = 'test'", myConnection)

myConnection.Open()

myCommand.ExecuteNonQuery()

myConnection.Close()

ExecuteReader

This method returns a SqlDataReader or OleDbReader object after executing the command. The Reader object contains the resultset returned from the database and can be used to bind a Bound control like the DataGrid as in the following example:

Dim myConnection As OleDbConnection

Dim dr As OleDbDataReader

Dim ConnStr As String

'Connect ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=Pubs; User ID=sa" myConnection = New OleDbConnection(ConnStr)

'command Dim myCommand As New OleDbCommand("p_authors", myconnection) myCommand.CommandType = CommandType.StoredProcedure

'Parameter object Dim objParam As OleDbParameter objParam = myCommand.Parameters.Add("State", OleDbType.VarChar, 10) objParam.Direction = ParameterDirection.Input objParam.Value = "CA"

'open the connection and execute the command myconnection.Open() 'ExecuteReader returns a Reader dr = myCommand.ExecuteReader()

'bind a grid DataGrid1.DataSource=dr DataGrid1.DataBind()

ExecuteScalar

The ExecuteScalar method is used to return a single result from the database (for example the count of the number of records in a table) as in the following examples:

Dim mycommand As New OleDbCommand($_$

"Select count(*) from masters", myConnection) myConnection.Open() myobject = myCommand.ExecuteScaler() myConnection.Close()

Action Queries with the Command Object

Action queries are SQL statements like Insert, Update, and Delete, which return no data. We use the Command object (OleDbCommand or SqlCommand) instead of the DataAdapter object to run such queries. A connection must be explicitly opened when using the Command object (whereas it is automatically opened when using the DataAdapter). The command is issued by calling an <u>ExecuteNonQuery</u> method, which returns the number of rows affected.

In the example that follows I will extend the MastersGrid discussed in the MastersGrid.aspx example and add functionality to insert, delete, and update a record in the Masters table. <u>Figure 3.2</u> shows what the result will look like.

Betessh Izsect Update Delete Code, value (code, dar)day code, catagory hype (closing operange) code value (code, catagory hype) 18 Wells Farge 605 A 10 0 19 Salary 702 I 20 0 22 Utiliser 704 E 10 0 25 Vasa 605 A 0 0 27 Discover Cand/605 A 0 0	
ode, valacio de daplavi code, cartegoritrigo elorango penna 18 W48 Fargo 605 A 10 0 19 Satary 702 I 20 0 22 Utilinies 704 E 10 0 25 Vaa 605 A 0 0 27 Discover Card 605 A 0 0	
Image: Solution of the second secon	
2. Utilises 704 E 10 0 5 Visa 605 A 0 0 7 Discover Card 605 A 0 0	
5 Visa 605 A 0 0 7 Discover Card 605 A 0 0	
7 Discover Card 605 A 0 0	
n bennen ennen he la la	



<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server">

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

if NOT (isPostBack)

rebind

end if

End Sub

Sub Show_Click(Sender As Object, E As EventArgs)

Message.Text = "Masters Table Displayed... "

ReBind

End Sub

Sub Insert_click(Sender As Object, E As EventArgs)

try

sql = "Insert into Masters(code_display,code_category,type)"

sql = sql + "Values ('test',701,'E')"

Dim mycommand As New OleDbCommand(sql,myConnection)

myConnection.Open()

myCommand.ExecuteNonQuery()

myConnection.Close()

catch myException as Exception

Response.Write("Exception: " + myException.ToString())

End try

rebind

Message.Text = "Inserted test record... "

End Sub

Sub Delete_click(Sender As Object, E As EventArgs)

sql = "delete from masters where code_display = 'test'"

Dim mycommand As New OleDbCommand(sql,myConnection)

myConnection.Open()

myCommand.ExecuteNonQuery()

myConnection.Close()

rebind

Message.Text = "Deleted all test records..."

End Sub

Sub Update_Click(Sender As Object, E As EventArgs)

try

Dim mycommand As New OleDbCommand(_

"UPDATE Masters Set Opening = 90 WHERE code_display = 'test'", _

myConnection)

myConnection.Open()

myCommand.ExecuteNonQuery()

myConnection.Close()

catch myException as Exception

Response.Write("Exception: " + myException.ToString())

End try

rebind

```
Message.Text = "Updated all test records: Set closing balance = 90...! "
```

End Sub

Sub ReBind()

SQL = "select * from Masters"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method to populate dataset

myCommand.Fill(ds, "Masters")

'Binding a Grid

DataGrid1.DataSource=ds.Tables("Masters").DefaultView

DataGrid1.DataBind()

End Sub

</script>

<body>

<h3>Action Queries</h3>

<form runat=server>

<asp:button text="Refresh" Onclick="Show_Click" runat=server/>

<asp:button text="Insert" Onclick="Insert_Click" runat=server/>

<asp:button text="Update" Onclick="Update_Click" runat=server/>

<asp:button text="Delete" Onclick="delete_Click" runat=server/>

<asp:label id="Message" runat=server/>

<asp:DataGrid id="DataGrid1" runat="server" />

</form>

</body>

</html>

You will note that the process of populating the DataSets and binding the grid is the same as in the MastersGrid.aspx example. However, I have moved the variable declarations outside the Page_Load event to give them global scope over the form. I have also added four buttons for the insert, update, delete, and refresh functionality. The if NOT (isPostBack) ... statement ensures that the grid is only loaded once (on page load). We have four events that fire when the appropriate button is clicked. The command syntax is straightforward. For example row deletion is achieved by the following code:

Sub Delete_click(Sender As Object, E As EventArgs)

sql = "delete from masters where code_display = 'test'"

Dim mycommand As New OleDbCommand(sql,myConnection)

myConnection.Open()

myCommand.ExecuteNonQuery()

myConnection.Close()

rebind

Message.Text = "Deleted all test records..."

End Sub

You pass the OleDbCommand a SQL string and an active connection, open the command, execute it, and then close it. Please note that it is very important to close the connection or else you may inadvertently exhaust the connection limit while waiting for the page instances to be garbage-collected.

Stored Procedures

In ADO.NET you can call stored procedures using the command object. You tell the command object the name of the stored procedure and then add a parameter for each input parameter required by the stored procedure. I also show you a "short-cut" method of calling stored procedures in MS SQL Server, where you take advantage of using the "Execute" keyword of T-SQL. Using this method, you can get away from the drudgery of populating the parameter collection.

I will use a simple stored procedure called <u>p_authors</u> that accepts a single input parameter called <code>@state</code> (you can find the code for the examples discussed in this section in the ...samples\StoredProcedure subdirectory on the book's Web site at <u>www.premierbooks.com/downloads.asp</u>). This stored procedure needs to be applied to the pubs database.

p authors

Create Procedure p_authors

@state varchar(10)

as

select * from authors where state = @state

Using the Parameters Collection

The command object exposes a Parameters collection that needs to be populated with each of the parameters expected by the stored procedure. In the example that follows, I show you how to call the stored procedure $\underline{p_authors}$ with a parameter of "CA". The result returned by the database will include all authors in the state of California. I will then bind this resultset to a DataGrid.

Parameters.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<head>

<H4>Using Parameters</H4>

<script language="VB" runat="server">

Sub Page_Load(Source As Object, E As EventArgs)

Dim myConnection As OleDbConnection

Dim dr As OleDbDataReader

Dim ConnStr As String

'Connect

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=Pubs; User ID=sa"

myConnection = New OleDbConnection(ConnStr)

'command

Dim myCommand As New OleDbCommand("p_authors", myconnection)

myCommand.CommandType = CommandType.StoredProcedure

'Parameter object

Dim objParam As OleDbParameter

objParam = myCommand.Parameters.Add("State", OleDbType.VarChar, 10)

objParam.Direction = ParameterDirection.Input

objParam.Value = "CA"

Try

'open the connection and execute the command

myconnection.Open()

'ExecuteReader returns a Reader

dr = myCommand.ExecuteReader()

Catch objError As Exception

'display error details here and stop execution on error

Exit Sub '

End Try

DataGrid1.DataSource=dr

DataGrid1.DataBind()

End Sub

</script>

</head>

<body>

<form runat=server>

<asp:DataGrid id="DataGrid1" runat="server" />

</form>

</body>

</html>

You will note that I have specified the <u>CommandType</u> as StoredProcedure after passing the name of the stored procedure as a parameter to the command object as follows:

Dim myCommand As New OleDbCommand("p_authors", myconnection)

myCommand.CommandType = CommandType.StoredProcedure
I then add a parameter to the Parameters collection as follows:

Dim objParam As OleDbParameter

objParam = myCommand.Parameters.Add("State", OleDbType.VarChar, 10)

objParam.Direction = ParameterDirection.Input

objParam.Value = "CA"

The Add method takes three arguments; the name of the parameter, its type, and optionally its size. The ParameterDirection property sets the direction of the parameter. This can be Input, Output, InputOutput, or ReturnValue. Finally the Value property is used to provide a value for the parameter.

The <u>ExecuteReader</u> method of the command object is used to execute the command. This returns a Reader object, which is then used to bind a DataGrid.

Using the Execute Keyword to Call Stored Procedures

The process of populating the Parameters collection of the command object described in the preceding section is quite code intensive. A short-cut way of calling stored procedures in MS SQL Server is available. You can make use of the Execute keyword of T-SQL to call a stored procedure.

Caution Note that this method will only work with MS SQL Server.

In the following example, I call the procedure p_authors using the Execute keyword: **Execute.aspx**

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<head>

<H4>Using execute Keyword</H4>

<script language="VB" runat="server">

Sub Page_Load(Source As Object, E As EventArgs)

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

'Connect

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=Pubs; User ID=sa"

myConnection = New OleDbConnection(ConnStr)

'DataSetCommand

SQL = "Execute p_authors 'CA'"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "Authors")

'Binding a Grid

DataGrid1.DataSource=ds.Tables("Authors").DefaultView

DataGrid1.DataBind()

End Sub

</script>

</head>

<body>

<form runat=server>

<asp:DataGrid id="DataGrid1" runat="server" />

</form>

</body>

</html>

Note that I make a call to the stored procedure p_authors and supply the required parameters with a single SQL statement using the Execute keyword as follows:

SQL = "Execute p_authors 'CA'" The code now is quite compact, as I did not have to write script to populate the Parameters collection of the command object.

DataViews

A DataView in ADO.NET is roughly equivalent to a database view. Different views can be applied to a DataTable existing in the DataSet. For example, one view could show all the rows in the table whereas another could show rows based on a selection criteria. List bound controls like DataGrids and DropDownLists then use these views as their datasource.

The Default View

Each DataTable has a default view assigned. In the MasterGrid.aspx example, the DataGrid is bound to the following default view:

DataGrid1.DataSource=ds.Tables("Masters").DefaultView

This default view is a view containing all the rows and columns of the DataTable.

Applying Filters to Views

In this example, I will populate a DataSet, which contains all the rows in the Groups table. I will then create a DataView which filters the Groups table in the DataSet for records having the criteria $code_value = 700$. In the process I will introduce the RowFilter and sort property of the DataView. Figure 3.3 shows a form with a button. If you click on this button, the appropriate filter is applied and the $code_display$ and the $code_value$ of all matching items are written out to the screen.



<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

DataView.aspx

<script language="VB" runat="server">

Sub readDs(Sender As Object, E As EventArgs)

Dim dv As DataView

Dim i As integer

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

SQL = "select * from groups "

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "groups")

dv = new DataView(ds.Tables("groups"))

'// Sort the view based on the code_display column

dv.Sort = "code_display"

'// Filter the dataview to only show customers with the lastname = Smith

```
dv.RowFilter = "code_value = '700'"
```

for i = 0 to dv.Count -1

Response.Write(dv(i)("code_display").ToString + " - " + dv(i)("code_value").ToString())

next

End Sub

</script>

<body>

```
<h3><font face="Verdana">DataView</font></h3>
```

<form runat=server>

<asp:button text="Read DataView" Onclick="readDs" runat=server/>

</form>

</body>

</html>

A connection is established to the database, and a DataSet is populated with all rows from the Groups table. A DataView is defined to hold all the rows from the DataSet. The sort command sorts the DataView according to the code_display. The RowFilter method filters the DataView using the criteria "code_value= '700'". The DataView is then iterated and the elements are displayed to the screen.

Reading the Rows and Columns Collection of a DataTable

In ADO a basic requirement was to read the "fields" collection of a recordset and display the field name of a database table alongside the value of the field. I will show you how this can be done in ADO.NET. I will read the Groups table and list out both the field name and the value of a record as shown in <u>Figure 3.4</u>.

nes gitup-Compute	o Colum	el cusionen unts	E free house	€]See it fogre Subreaun Stes	@]Wirdlan Media	@]windows
Jaca Tabi	e coluii	IIIS & K	ows			
id	R					
code_display	Sales account					
code_value	700					
code_category	7					
type	1					

Figure 3.4: Reading rows and columns collection of a DataTable. Collection.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server">

Sub Page_Load(sender As Object, e As EventArgs)

Dim dv As DataView

Dim i As integer

Dim myConnection As OleDbConnection

Dim myCommand As OledbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

SQL = "select * from groups where code_value = 700 "

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "groups")

dv = new DataView(ds.Tables("groups"))

Dim t As DataTable

t = dv.Table

Dim r As DataRow

Dim c As DataColumn

Dim _cell as TableCell

Dim _row as TableRow

table = New Table()

Controls.Add(table)

For Each r in t.Rows

For Each c in t.Columns

_row = new TableRow() '

'Label

_cell = new TableCell() '

_cell.Controls.Add(new LiteralControl(c.ToString))

_row.Cells.Add(_cell) '

'Value

_cell = new TableCell() '

_cell.Controls.Add(new LiteralControl(r(c).ToString))

_row.Cells.Add(_cell) '

Table.Rows.Add(_row) '

Next c

Next r

End Sub

</script>

<form runat = "server">

<h2> DataTable Columns & Rows </h2>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<asp:Table id="Table" Font-Name="Verdana" Font-Size="8pt" CellPadding=5 CellSpacing=0 BorderColor="black"

BorderWidth="1" Gridlines="Both" runat="server"/>

</body>

</form>

</html>

The Groups table in the DataSet is populated with the SQL query "select * from groups where code_value = 700 ". A DataView is created on the Groups table as follows:

dv = new DataView(ds.Tables("groups"))

This DataView is assigned to a DataTable. I then have two loops. The outer loop iterates the row collection of the DataTable and the inner loop iterates the column collection. The basic loop is as follows (I have removed all formatting elements like Table, TableRow, and TableCell, so that I can explain better):

Dim t As DataTable

t = dv.Table

Dim r As DataRow

Dim c As DataColumn

Dim cell As TableCell

Dim row As DataRow

For Each r in t.Rows

For Each c in t.Columns

response.write(c.ToString)

response.write(r(c).ToString)

Next c

Next r

I am dynamically creating the Table (tag), the TableRow (), and the TableCell (). Thus an opening Table tag is created as follows:

table = New Table() 'supplies tag

Controls.Add(table)

I want to display the column name as a non-editable label. I use the following syntax:

_cell = new TableCell() 'Supplies tag

_cell.Controls.Add(new LiteralControl(c.ToString))

_row.Cells.Add(_cell) 'supplies

The TableCell() method takes care of supplying the and tags. c is the field name and it is converted to a string using the ToString method. LiteralControl adds a label. Thus the field name is displayed as a label.

I want to display the value of each field in an editable textbox. This is achieved by the following code:

_cell = new TableCell() '

Dim Box As New TextBox

Box.Text = r(c).ToString

_cell.Controls.Add(box)

_row.Cells.Add(_cell) '

The new TableCell() and the Add(_Cell) provide the opening and closing table data tags (i.e. and). I assign the column value to a textbox and add it to the controls collection.

The DataReader

The DataSet provides a disconnect means of access to a datasource. At times we might want a "quick and dirty" means of accessing a datasource. In such cases, a DataReader can be used.

A DataReader supplies a read-only, forward-only data stream and like the legacy ADO recordset, stays connected to the datasource. It can be used to return a recordset or execute action queries (like update, insert, and delete) which do not return any data. It holds one row in memory at a time as opposed to a DataSet, which holds a complete table in memory. Using a DataSet can be an issue when large tables are loaded in memory. If there are multiple users accessing the same machine at the same time, this can lead to a serious memory drain. In such situations a DataReader should be used instead of the DataSet.

A DataReader provides a simple method of iterating through the query. In the following example (see <u>Figure 3.5</u>), I populate a DropDownList by iterating through the Groups table using the DataReader.



Figure 3.5: DataReader. DataReader.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server">

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

if NOT (isPostBack)

FillList

End if

End Sub

Sub FillList()

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

Dim dbRead AS OleDbDataReader

Dim dbComm AS OleDbCommand

SQL = "Select * from groups ORDER BY code_value"

dbComm = New OleDbCommand(SQL,myConnection)

myconnection.Open()

dbRead=dbComm.ExecuteReader()

While dbRead.Read()

ddList.items.add(New ListItem(dbRead.Item("code_display")))

End While

End Sub

```
</script>
```

<body>

<h3>DropDownList & Reader </h3>

<form runat=server>

<asp:DropDownList id="ddlist" runat="server"

DataTextField = "code_display"/>

</form>

</body>

The DataReader uses the OleDbCommand to populate the command object. After the execute method is run, we can iterate dbRead (the DataReader) to populate the DropDownList.

Data Relation

As mentioned earlier in this chapter, you can set up relationships between tables in the DataSet. These relationships are akin to the primary-foreign key relationships, which exist in a database. The advantage of defining a relationship is that you can now navigate the relationship instead of navigating in a sequential manner as done previously in ADO. In this style of navigation, a master record is first selected, and then based on the relation key the row in the secondary table is accessed. Processing in the secondary table continues until all secondary records with the same primary key are processed. After this, the control moves back to the master table and another row is processed in a similar manner. Let me explain this with an example.

DataRelation.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server">

Sub readDs(Sender As Object, E As EventArgs)

Dim dv As DataView

Dim i As integer

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=pubs;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

'Populate authors table

SQL = "select * from authors "

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "authors")

'Populate TitlesAuthor

SQL = "select * from titleAuthor"

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "titleauthor")

'Define a relation based on the au_id

'dc1 is the primary, dc2 is the secondary table

Dim dc1 As DataColumn

Dim dc2 As DataColumn

dc1 = ds.Tables("Authors").Columns("au_id")

dc2 = ds.Tables("titleauthor").Columns("au_id")

Dim dr As DataRelation

dr = New DataRelation("vRelation", dc1, dc2)

ds.Relations.Add(dr)

'Loop thru the relation

Dim child() As datarow

Dim r As datarow

for each r in ds.Tables("Authors").Rows

child = r.GetChildRows(ds.Relations("vRelation"))

for i = 0 to UBound(child)

Response.Write(child(i)("au_id").ToString)

next

next

End Sub

</script>

<body>

<h3>DataView</h3>

<form runat=server>

<asp:button text="Read Relation" Onclick="readDs" runat=server/>

</form>

</body>

</html>

In this example, I populate the DataSet ds with two tables from the Pubs database. These are the author's table (the primary table) and the title Author table (the child table). I then define a relationship (vRelation) between the two tables, based on the au_id key in both tables. I then loop through all the records in the relationship. The external loop iterates through the author's table, one row at a time. It picks up a row and then loops through all child records in the titleauthor table, which have the same au_id. This is done by the internal loop.

For example, you can use the select method of the DataSet to filter records:

Dim child() As DataRow = workTable.Select("au_id like 'A%'").

Like a database, the DataSet supports unique and cascading constraints. You can enclose your code within a BeginEdit and EndEdit block. This in effect defers constraint validation until the EndEdit method is called. Three values are stored for each row. These are the original, current, and proposed values. The proposed value is the intermediate value between the BeginEdit and EndEdit block. The currentvalue becomes the originalvalue once the AcceptChanges method is called. Finally, the RejectChanges method drops changes to the DataSet.

Summary

This chapter looked at working with ADO.NET. This new data access technology involves working with DataSets and Managed Providers. I introduced data binding with ADO.NET, and showed how action queries could be performed using ADO.NET. DataViews and DataRelations were explained in detail. The <u>next chapter</u> extends this knowledge, and shows you how to bind controls using ADO.NET.

Chapter 4: Data Binding

Controls can be bound to a datasource, much like Visual Basic Bound controls. Thus controls like the DropDownList, CheckBoxList, and RadioButtonList can be bound to a DataSet. ASP.NET has certain controls that render HTML based on repetitive data. These controls are collectively referred to as "List Bound Controls." The controls included in this category are the DataRepeater, the DataList, and the DataGrid. Developers can apply various styles and set properties in an XSL template-like style. There is a header template, a footer template, and an item template. As the names imply, the header and footer templates control the header and footer sections respectively. The item template is used to control repetitive data. In a DataGrid, the item template is applied to columns. The item template can be further fine-tuned by alternating it with a separator template. You can use these templates to apply different colors to odd and even rows. The DataRepeater does not have editing capabilities, though both the DataList and the DataGrid do. In addition, the DataGrid has advanced paging and sorting capabilities.

Binding Controls

You can bind controls like the CheckBoxList, the RadioButtonList, the ListBox, and the DropDownList to DataSets. Binding is as simple as specifying the datasource and binding the control using the DataBind method. In the following example, I show you how to bind a ListBox, a DropDownList, and a set of RadioButtons (a RadioButtonList) to the data from the Groups table. The result appears in Figure 4.1.



Figure 4.1: Binding "selection" controls. DataBind.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server">

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

if NOT (isPostBack)

bind end if End Sub Sub Bind() 'DataSetCommand SQL = "select * from Groups" myCommand = New OleDbDataAdapter(SQL, myConnection) 'use Fill method of DataSetCommand to populate dataset myCommand.Fill(ds, "Groups") list1.DataSource=ds.Tables("Groups").DefaultView list1.DataBind() rb.DataSource=ds.Tables("Groups").DefaultView rb.DataBind() dl.DataSource=ds.Tables("Groups").DefaultView dl.DataBind() End Sub Sub SubmitBtn_Click(sender As Object, e As EventArgs) Dim s As string s = "---Selected List Item :" + list1.SelectedItem.Text s = s + "-----Selected DropDownList : " + dl.SelectedItem.Text s = s + "----Selected RadioButton : " + rb.SelectedItem.Text Label1.Text = s End Sub </script>

```
<body>
```

<h3>Binding Controls </h3>

<form runat=server>

<asp:ListBox id="List1" DataTextField = "code_display" DataValueField = "code_value" runat="server"/>

<hr>

<ASP:radiobuttonList

repeatcolumns="4" repeatdirection="horizontal" repeatlayout="table"

id="rb" datatextfield="code_display" DataValueField="code_value" runat="server"/>

<hr>

<asp:DropDownList id="dl" DataTextField = "code_display" DataValueField = "code_value" runat="server"/>

<hr>

<asp:button Text="Submit" OnClick="SubmitBtn_Click" runat=server/>

<asp:Label id=Label1 font-name="Verdana" font-size="10pt" runat="server" />

</form>

</body>

</html>

Binding involves specifying the DataSource and then using the DataBind method to perform the actual bind, as in the following example:

list1.DataSource=ds.Tables("Groups").DefaultView

list1.DataBind()

CheckBoxLists and RadioButtonLists have a RepeatColumns and a

RepeatDirection property. RepeatColumns can be used to assign the number of columns repeated in the direction specified by the RepeatDirection property. In the preceding example I have set the RepeatColumns property to 4 and the RepeatDirection property to be horizontal.

In the SubmitBtn_Click event, I use the SelectedItem.text property of each of these selection controls (the ListBox, the DropDownList, and the RadioButtonList) to display the item selected by the user.

Each of these controls has an AutoPostBack property, which can be set as true or false, such as in the following:

<asp:ListBox id="List1" DataTextField = "code_display" DataValueField = "code_value" runat="server" AutoPostBack = 'true" />

Setting this property to true will cause a post back to the server each time that a value in the control is changed. The "if NOT (isPostBack)" statement ensures that the controls are only bound once at the time of page_load.

The DataRepeater

The DataRepeater is used to render HTML for repeating data. It is completely template driven and the following templates can be set for it:

- ItemTemplate
- AlternatingItemTemplate
- SeparatorTemplate
- HeaderTemplate
- FooterTemplate

The ItemTemplate is the only required template and it defines the content and layout of the list. The AlternatingItemTemplate defines the content and layout of alternating items. The SeperatorTemplate is for items between the items and alternating items. Finally, the HeaderTemplate and FooterTemplate determine the rendering of the header and footer of the list, respectively. The DataRepeater has no built-in styles and all HTML elements must be explicitly specified in the various templates. For example, to give a tabular look, you would define the table tab in the HeaderTemplate, the closing table tag in the FooterTemplate and the table row, and table data tags in the ItemTemplate (and/or in the

AlternatingItemTemplate). In <u>Figure 4.2</u>, I render the Groups table in a tabular format using the DataRepeater.





<%@ Import Namespace="System.Data" %>

```
<%@ Import Namespace="System.Data.OleDb" %>
```

<html>

<script language="VB" runat="server">

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

if NOT (isPostBack)

rebind end if End Sub Sub ReBind() 'DataSetCommand SQL = "select * from Groups" myCommand = New OleDbDataAdapter(SQL, myConnection) 'use Fill method of DataSetCommand to populate dataset myCommand.Fill(ds, "Groups") 'Binding a Grid DataGrid1.DataSource=ds.Tables("Groups").DefaultView DataGrid1.DataBind() End Sub </script> <body> <h3> Data Repeater</h3> <form runat=server> <asp:Repeater id="DataGrid1" runat="server"> <HeaderTemplate>

```
Type</b>
```

</HeaderTemplate>

<ItemTemplate>

<%# Container.DataItem("code_display") %>

<%# Container.DataItem("code_category") %>

```
<%# Container.DataItem("type") %>
```

</ltemTemplate>

<AlternatingItemTemplate>

<%# Container.DataItem("code_display") %>

<%# Container.DataItem("code_category") %>

<%# Container.DataItem("type") %>

</AlternatingItemTemplate>

<FooterTemplate>

</FooterTemplate>

</asp:Repeater>

</form>

</body>

</html>

This sample populates a DataSet with records from the Groups table. The opening table tag is in the HeaderTemplate, and the closing table tag in the FooterTemplate. The table data and table row tags are applied both in the ItemTemplate and the AlternatingItemTemplate. Finally, every other row has a silver background due to the style specified in the AlternatingItemTemplate.

DataGrid

The DataGrid control is used to create attractive, tabular layouts of data associated with it. With the DataGrid control you can specify styles for the header row, the footer row, the item rows the alternating rows and also create column level templates. The DataGrid supports advanced features like in-line editing, sorting, and paging.

To introduce the DataGrid, I will build a form to add, update, and delete records in the Masters table. I will then present a sequence of forms, each of which presents a new facet of the DataGrid. In the end, I will have a form with editing capabilities. I will continue working on this form in <u>Project 1</u>, and at the end I will have a full-fledged Maintenance Form.

A Basic Grid

In the DataBind.aspx example, I showed you how to bind to a DataGrid and present the results. The following statements in the Page_Load event do this:

DataGrid1.DataSource=ds.Tables("Masters").DefaultView

DataGrid1.DataBind()

The DataBind method causes the DataGrid to refresh its data. Bound controls will be populated with data only when this method is called. This gives control to the developer as he can now refresh data as needed, preventing unnecessary datasource access. Typically, this method is called at Page_Load and then when it is required to refresh the DataGrid. Calling this method at the page level (page.DataBind() or just DataBind()) will cause all data binding expressions on the page to be evaluated.

You will observe that the resulting DataGrid is quite dull, so let's embellish it by applying styles.

The Masters Grid with Style

Figure 4.3 shows what the Masters grid will look like after the application of styles.

100000	Search a stra	Buchmark	ateres	the tagement to Ve		16 P 10
Z/ - S IS NI	Da Control II.	Chine and		ang a come ano re	e	
- tax • 0	C C Cosean Creve	CH OH	story L(yea		
oguese (45) yeah (Nora)	hods/AspAietBuck/Chapter30thap3_4	anpies/Me	tees: Jeep			• c ² in
nis @)Casconiae.ini	s @reentonial @lteachtr	one Subre	son Stes	@] Wincows Media	e @windows	
hart of Accounts						
Account	Greep	Type	Open	ing Cleang		
falls Fargs	8-ank accounts	3,	8	1.0		
siony	Inceme acceunt	1		20		
tilities.	Expenditure account.	Ε.		1.0		
58	Bank accounts	4		0		
iscover Card	Bank accounts	5.		0		

Figure 4.3: Master1 DataGrid.

Our code is in Masters1.aspx, which has a Code Behind file called Masters1.vb. Following the aspx form is an explanation.



<html>

<head>

<title>Masters DataGrid 1</title>

</head>

<body>

<form runat=server>

Chart of Accounts:

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt">

<Columns>

<asp:BoundColumn HeaderText="Account" DataField="code_display">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Group" DataField="category">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Type" DataField="type">

<HeaderStyle Width="50px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Opening" DataField="opening">

<HeaderStyle Width="50px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Closing" DataField="closing">

<HeaderStyle Width="50px">

</HeaderStyle>

</asp:BoundColumn>

</Columns>

<HeaderStyle BackColor="DarkRed" ForeColor="White" Font-Bold="true">

</HeaderStyle>

<ItemStyle ForeColor="DarkSlateBlue">

</ltemStyle>

<AlternatingItemStyle BackColor="Beige"/>

</asp:DataGrid>

</form>

</body>

</html>

The script of the Code Behind file, Masters1.vb is as follows: Masters1.vb

Imports System

Imports System.Collections

Imports System.Text

Imports System.Data

Imports System.Data.OleDb

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class BaseClass

Inherits System.Web.UI.Page

Protected Grid1 as DataGrid

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

if NOT (isPostBack)

rebind

end if

End Sub

Sub ReBind()

'DataSetCommand

SQL = "select m.*, g.code_display as category "

SQL = SQL + "from masters m, groups g "

SQL = SQL + " where m.code_category = g.code_value"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "masters")

'Binding a Grid

Grid1.DataSource=ds.Tables("masters").DefaultView

Grid1.DataBind()

End Sub

End Class

Let's dissect the code. The first thing you will note is that I told the DataGrid not to automatically generate columns by turning off the AutoGenerate attribute. I then set the Font, BackColor, ForeColor, CellPadding, and CellSpacing properties. Next, I set the property tags of the DataGrid to specify the columns to be displayed. You will note that this is an XML-like tag setting. Finally, note that the DataGrid has a HeaderStyle with which I customize the header row, an ItemStyle which is used to render the ForeColor of all elements to DarkSlateBlue, and an AlternatingItemStyle which I use to render every other row's BackColor to Beige.

The Editable Masters Form

I now add editing capabilities to the grid. The grid will now have a link called "edit," which will take you to the edit mode when clicked on. <u>Figure 4.4</u> shows what the DataGrid looks like at this stage.

	El sel (locaros	SAspAintSock, Over	¥300M	3"read	er/Meter	HS WDI				- C'
nis 创	Cestoniae Links	e]nee Homal 🐇]Sead	Engre	Subnesso	in Stes	C WINCOWS MR	ode @]win	lovs	
hart	of Acco	unts								
lick to	Edit Account	# Name	Grou	а Тур	e Openi	ng Clas	ing			
45	20	Wells Kergo	535	A.	0	10				
31	29	Salary	732	1	0	20				
18	22	Utilitet	734	1	0	10				
8	26	Visa	6.75	4	0	0				
<u>88</u>	27	Discover Card	0.05	4	0	0				

Figure 4.4: Masters2 DataGrid.

Two new links, with the captions "OK" and "Cancel," will appear at this stage. We can either accept the updates by clicking "OK," or leave the data as it was by selecting "Cancel." Figure 4.5 shows what the form will look like in edit mode.

Chart of Accounts		in sides all wandows media	@]windows	-1 (* s
lick to Edit Account Name	Greep	Type	Opening	Closing
dt 10 Welle Fe	argo 605	A	0	1.0
dž 19 Colory	702	1	0	29
di 22 Utilites	704	ε	0	10
K Canual 20 Mag	605		1 A	
dt 27 Eiscove		<u>^</u>	llo.	9

Figure 4.5: Masters2 DataGrid in Edit Mode. Masters2.aspx

<%@Page Language="VB" Inherits="BaseClass" Src="masters2.vb" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server">

Sub Grid1_Update(sender As Object, e As DataGridCommandEventArgs)

Dim sql As string

Dim code_display As String

Dim code_category As String

Dim type As String

Dim opening As String

Dim closing As String

Dim myTextBox As TextBox

'This is the key value: Retrieved from the DataKey, since it's a read only field

Dim code_value as string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString

myTextBox = E.Item.FindControl("edit_name")

code_display = mytextbox.text

myTextBox = E.Item.FindControl("edit_group")

code_category = mytextbox.text

myTextBox = E.Item.FindControl("edit_type")

type = mytextbox.text

myTextBox = E.Item.FindControl("edit_opening")

opening = mytextbox.text

myTextBox = E.Item.FindControl("edit_closing")

closing = mytextbox.text

'Now execute stored procedure

sql = "Execute p_masters " + code_value + ", '" + code_display + " ',"

sql = sql + code_category + ", " + type +" ," + opening + "," + closing

RunSql(sql)

Grid1.EditItemIndex = -1

ReBind()

End Sub

</script>

<body style="font: 10pt verdana">

<form runat="server">

<h3>Chart of Accounts </h3>

<asp:Label id="Message" runat="server"/>

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

OnEditCommand="Grid1_Edit"

OnCancelCommand="Grid1_Cancel"

OnUpdateCommand="Grid1_Update"

DataKeyField="code_value">

<Columns>

<asp:EditCommandColumn

EditText="Edit"

CancelText="Cancel"

UpdateText="OK"

ItemStyle-Wrap="false"

HeaderText="Click to Edit"

HeaderStyle-Wrap="false"/>

<asp:BoundColumn HeaderText="Account #" ReadOnly="true" DataField="code_value"/>

<asp:TemplateColumn HeaderText="Name" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("code_display") %>' runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_name" Text='<%# Container.DataItem("code_display") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Group" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("code_category") %>' runat="server"/> </ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_group" Text='<%# Container.DataItem("code_category") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Type" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("type") %>' runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_type" BorderStyle="None" Readonly="True" Text='<%#
Container.DataItem("type")</pre>

%>'runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Opening" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("opening") %>' runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_opening" Text='<%# Container.DataItem("opening") %>' runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Closing" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("closing") %>' runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_closing" BorderStyle="None" Readonly="True"

Text='<%# Container.DataItem("closing") %>' runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

</Columns>

<HeaderStyle BackColor="DarkRed" ForeColor="White" Font-Bold="true">

</HeaderStyle>

<ItemStyle ForeColor="DarkSlateBlue">

</ltemStyle>

<AlternatingItemStyle BackColor="Beige">

</AlternatingItemStyle>

</asp:DataGrid>

</form>

</body>

</html>

Masters2.vb is the Code Behind file for this form. Here is its listing: Masters2.vb

Option Strict Off

Imports System

Imports System.Collections

Imports System.Text

Imports System.Data

Imports System.Data.OleDb

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class BaseClass
Inherits System.Web.UI.Page

Protected Grid1 as DataGrid

Protected Message as label

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

```
ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"
```

myConnection = New OleDbConnection(ConnStr)

if NOT (isPostBack)

rebind

end if

End Sub

Sub ReBind()

SQL = "select m.*, g.code_display as category "

SQL = SQL + "from masters m, groups g "

SQL = SQL + " where m.code_category = g.code_value"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "masters")

'Binding a Grid

Grid1.DataSource=ds.Tables("masters").DefaultView

Grid1.DataBind()

End Sub

------ New Events (additions to Masters1.vb)------

Sub Grid1_Edit(Sender As Object, E As DataGridCommandEventArgs)

Grid1.EditItemIndex = E.Item.ItemIndex

ReBind()

End Sub

Sub Grid1_Cancel(Sender As Object, E As DataGridCommandEventArgs)

Grid1.EditItemIndex = -1

ReBind()

End Sub

Sub RunSql(sql as string)

'This is a placeholder for the functionality we will code in Masters3.vb

response.write(sql)

End Sub

End Class

I need to explain quite a number of things here, so I will run down the code contained in the files Masters2.aspx and Masters2.vb one step at a time.

Look at the tags that create the DataGrid in the aspx file. The first column that I have specified is the EditCommandColumn. This is an ASP.NET generated column and it creates the "Edit" link button. When in Edit mode it creates the "OK" and "Cancel" buttons. There are three events (OnEditCommand, OnCancelCommand,

OnUpdateCommand) that correspond to these buttons. I have coded three functions that get fired when these events get executed (Grid1_edit, Grid1_Cancel, Grid1_update).

Template columns are used to organize a DataGrid's columns , much like an XSL template. It allows the developer to set the properties for the column. Each column can have two templates: An ItemTemplate and an EditItemTemplate. An

ItemTemplate displays the value of the column in a read-only manner. When the grid goes into the Edit mode, the EditItemTemplate is displayed, and you can change the column value. Think of the ItemTemplate and EditItemTemplate as two separate controls (with separate ids). The following code shows what the "Name" column looks like:

<asp:TemplateColumn HeaderText="Name" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("code_display") %>' runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_name" Text='<%# Container.DataItem("code_display") %>' runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

The Container means the parent control, which in this case is the grid. I am binding the label to the code_display DataItem of the grid.

The grid needs to be told which row is being edited. This is done by setting the EditItemIndex property of the DataGrid to the index of the button that was clicked as in the Grid1_edit event. To cancel the editing just set the EditItemIndex to -1 as in the Grid1_cancel event. Note that in each case I have to rebind the DataGrid for the changes to take effect.

The Grid1_update event gets fired when the "OK" button is clicked. I have written a stored procedure, p_masters, which takes care of the business of adding and updating rows. I will discuss this procedure in detail in <u>Chapter 5</u>, "<u>Input Validation.</u>" For the moment, it is sufficient to know that the call syntax is as follows:

Execute p_masters @code_value, @code_display, @code_category, @type, @opening, @closing.

When I pass the procedure a code_value (the primary key), it updates the record with that code_value with the new values. If I pass it a null code_value, it inserts a new record.

Using a stored procedure is a great way to encapsulate the insert/update functionality. Your form becomes very lean; it is now only concerned with extracting appropriate values and passing them onto the stored procedure. The stored procedure can do validation, multiple table updates, and much more.

I will not actually call the procedure but I will build the syntax and write it to the screen. Also note that I am only dealing with the "update" mode. I will incorporate the record addition and deletion functionality and work with real calls to the procedure. In the "DataGrid Tag" I specified that the DataKeyField was equal to the "code_value". This is the primary key and it is extracted as follows:

Dim code_value as string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString In the EditItemTemplate of each column, I gave each column a unique id. Thus, the name column had an id = edit_name when in Edit mode. I find this control using the FindControl method and assign it to a textbox, where I can access its text property.

Dim myTextBox as textbox

Dim code_value as string

myTextBox = E.Item.FindControl("edit_name")

code_display = mytextbox.text

To build the stored procedure call, I build the following string:

sql = "Execute p_masters " + code_value + ", '" + code_display + " ',"

sql = sql + code_category + ", " + type +" ," + opening + "," + closing

Using the Execute keyword I am telling our database to execute a SQL command (specified in the sql string). I find this syntax very convenient to use. You could call the stored procedures using the command object and setting the parameters to be passed to the stored procedure. However, I find that I need to write much longer code that way, as I have to write a line of code for each parameter. Here, I can make the procedure call in two lines of code.

Caution

The use of the Execute keyword to run a stored procedure will only work with MS SQL Server databases. In order to call a stored procedure in other databases, you need to use the Command object, set its CommandType property to StoredProcedure and use the Parameters property to access input and output parameters and return values.

Sorting and Paging

The DataGrid includes features that allow you to set up sorting and paging functionality. When the sorting functionality is enabled, links appear under the column header names. When you click on a column link, that column sorts the grid.

The paging functionality of the DataGrid allows you to set the number of records that can be displayed per page. Users can then navigate to different recordsets by clicking on the paging links that appear at the bottom of the DataGrid. These links can appear as numeric links or VCR-type "next" and "previous" buttons.

I will base my discussion on an example I have developed, the code of which is contained in the file PagingSorting.aspx and its associated Code Behind file PagingSorting.vb. These files can be found in the samples folder for this chapter on the book's Web site at www.premierpressbooks.com/downloads.asp.

Sorting

The DataGrid allows you to sort the columns by clicking on a link below the columns. Setting the AllowSorting property to true triggers this built-in mechanism and it is as follows:

<asp:DataGrid id="Grid1" runat="server"

AllowSorting="true"

OnSortCommand="MyDataGrid_Sort">

When this property is set to true, the DataGrid renders the column captions with a LinkButton. If you now click on a column, the OnSortEvent is fired. This event contains the following code:

Sub MyDataGrid_Sort(sender As Object, e As DataGridSortCommandEventArgs)

SortField = e.SortField

ReBind

End Sub

The SortField variable is a Public (string) variable that holds the name of the column by which the DataGrid is to be sorted. It is first set in the Page_Load event and later whenever the user clicks on a sortable column. The Page_Load setting is as follows:

Public SortField As String

Sub Page_Load(Source As Object, E As EventArgs)

If NOT (isPostBack)

If SortField = "" Then

SortField = "code_display"

End If

ReBind

End If

End Sub

The rebind function uses the SortField to sort the DataView to which the DataGrid is bound. It refreshes the DataGrid to reflect the rows sorted by the new sort field in the sub called Rebind as follows:

Sub ReBind()

'DataSetCommand

SQL = "select m.*, g.code_display as category "

SQL = SQL + "from masters m, groups g "

SQL = SQL + " where m.code_category = g.code_value"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset myCommand.Fill(ds, "masters") 'Sort accounding to sortField Dim dv2 As DataView dv2 = ds.Tables("masters").DefaultView dv2.Sort = SortField Grid1.DataSource= dv2 Grid1.DataBind() End Sub

You need to set the SortField property in the column templates. For example for the code_display column to participate in sorting, you have to set the template as follows:

<asp:BoundColumn HeaderText="Account" DataField="code_display"

SortExpression="code_display">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

Paging in DataGrid

The DataGrid has a built-in pager control which displays a user-defined number of pages per page and also numeric or "next/previous" buttons at the bottom of the DataGrid. Clicking on these links displays the next set of pages and so on. To enable paging you set a number of properties as follows:

<asp:DataGrid id="Grid1" runat="server"

AllowPaging="True"

PageSize="5"

PagerrStyle-Mode="NumericPagesi"

PagerrStyle-HorizontalAlign="Right"

PagerrStyle-NextPageTText="Next"

PagerrStyle-PrevPageTText="Prev"

OnPageIIndexChanged="MyDataGrid_Page"&>

The AllowPaging property must be set to true to enable paging. The PageSize property sets the number of records per page. A PageSize of 5 implies that only five records per page will be shown. If you leave out the PagerStyle-

Mode="NumericPages" property then instead of numeric links at the bottom you get two links; next and previous. The PagerStyle-NextPageText and the PagerStyle-PrevPageText properties are descriptive captions for these two links and they can be any text you want.

You are required to code one event. This is the OnPageIndexChanged event, which fires off the MyDataGrid_Page event. You simply call the rebind function in this event as follows:

Sub MyDataGrid_Page(sender As Object, e As DataGridPageChangedEventArgs)

ReBind

End Sub

Here is the compete source listing: PagingSorting.aspx <%@Page Language="VB" Inherits="BaseClass" Src="PagingSorting.vb" %>

<html>

<head>

<title>Masters DataGrid 1</title>

</head>

<body>

<form runat=server>

Sorting and Paging

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

AllowPaging="True"

PageSize="5"

PagerStyle-Mode="NumericPages"

PagerStyle-HorizontalAlign="Right"

PagerStyle-NextPageText="Next"

PagerStyle-PrevPageText="Prev"

OnPageIndexChanged="MyDataGrid_Page"

AllowSorting="true"

OnSortCommand="MyDataGrid_Sort"

>

<Columns>

<asp:BoundColumn HeaderText="Account" DataField="code_display" SortExpression="code_display">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Group" DataField="category" SortExpression="category">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Type" DataField="type">

<HeaderStyle Width="50px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Opening" DataField="opening">

<HeaderStyle Width="50px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Closing" DataField="closing">

<HeaderStyle Width="50px">

</HeaderStyle>

</asp:BoundColumn>

</Columns>

<HeaderStyle BackColor="DarkRed" ForeColor="White" Font-Bold="true">

</HeaderStyle>

<ItemStyle ForeColor="DarkSlateBlue">

</ltemStyle>

<AlternatingItemStyle BackColor="Beige"/>

</asp:DataGrid>

</form>

</body>

</html>

The Code Behind for this form is as follows: **PagingSorting.vb**

Imports System

Imports System.Collections

Imports System.Text

Imports System.Data

Imports System.Data.OleDb

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class BaseClass

Inherits System.Web.UI.Page

Protected Grid1 as DataGrid

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Public SortField As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

if NOT (isPostBack)

If SortField = "" Then

SortField = "code_display"

End If

rebind

end if

End Sub

Sub ReBind()

'DataSetCommand

SQL = "select m.*, g.code_display as category "

SQL = SQL + "from masters m, groups g "

SQL = SQL + " where m.code_category = g.code_value"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "masters")

'Sort accounding to sortField

Dim dv2 As DataView

dv2 = ds.Tables("masters").DefaultView

dv2.Sort = SortField

Grid1.DataSource= dv2

Grid1.DataBind()

End Sub

Sub MyDataGrid_Page(sender As Object, e As DataGridPageChangedEventArgs)

Grid1.CurrentPageIndex = e.NewPageIndex

ReBind

End Sub

Sub MyDataGrid_Sort(sender As Object, e As DataGridSortCommandEventArgs)

SortField = e.SortExpression

ReBind

End Sub

End Class

The DataList

The DataList is completely template-driven. The following templates can be applied to the DataList:

ItemTemplate

AlternatingItemTemplate

SeparatorTemplate

- SelectedItemTemplate
- EditItemTemplate
- HeaderTemplate
- FooterTemplate

Using these templates you can apply various styles to the header, footer, items, and alternating items. The templates work just as they do in the Data Repeater. The difference is that you can define an "EditItemTemplate" at item level as in a DataGrid. This gives editing capabilities to the DataList. Where the DataList differs from the DataGrid is that you can specify the "RepeatDirection" and "RepeatColumns" properties. With the RepeatDirection property you can either render the datasource in a horizontal or vertical direction. With the RepeatColumns property you can control the number of columns that are rendered in a specified direction. In the example that follows, I build a web form, which derives its data from the Groups table. It renders columns, three across, and has editing functionality. Figure 4.6 shows what it looks like. Figure 4.7 shows what the DataList looks like in Edit mode.

dess @ tery (nonhost/Academics) is @ Castoneeunis @ Preentone is of Castoneeunis @ Preentone Castoneeunit def River a account def River a account def River a account def River a	Chapter/Shrup/Comped/on.ped/at.aspo al @Search.tradne.Subresson.tites @Search Edit Lang term loans . Edit Lang term loans . Edit Concernante Edit Decemment	ousmade @livindons	•
Contraction of the second	dig Lang term lonns dig Lang term lonns dig Lang term lonns dig bin-digtantia	ovs Heda 🕡 Windows	
ane di Captel seconn di Ford esete di Ford esete di Forde seconti di Second leans di Forde (seconti) di Forde (seconti)	Edd Long term loans		
aanse dat Cepitel secount dat Exved assets dat Exvenue accounts dat Exvenue accounts dat Experies dat Experies (at Deceste Caresta)	Edit Lang term loans Edit Divestmenta Edit Drandt/divesons		
<u>dat Capital account</u> <u>dat Rived assets</u> <u>dat Revenue accounts</u> <u>dat Beourned Ioono</u> <u>dat Provesions</u> <u>dat Decosits (assets)</u>	Edit Leng term loans Edit Drivestments Edit Branch/SiveLons		
Lat Fixed essets Lat Revenue accounts Lat Secured Ioana Lat Provisions Lat Deposits (essets)	Edit Branch/divisions	EEE CUPYERE INDUIDES	_
dit Beoured kons dit Beoured kons dit Provisions dit Deposite (assets)	Loc branch/divisions	Ede Ourvent assets	
dit Secured Roma dit Provisions dit Deccelta (assetta)	a feature and the second second	Ldc Released and surgius	_
dit Deposite (assota)	Los Cristoures reading	Life Dates taxes payable (ba)	_
AND DESCRIPTION OF A DESCRIPTION OF	Line Scholary (restors	Life Grades debters	
dt Cash in band	Edit Bank accounts	147 Shies account	-
idit Purshase account	Edit Decome account	Edit Outles and taxes and faith	
dt Expenditure account	Edit Advances - excise a/os	Edit Expenses (direct)	-
dt Expenses (indirect)	Sector sector	and the second second	
	anti (2.11800ando Latentingto.com anti (2.1600ata (3.1600) (2.6) (2.	needs internet ingener rectio varios	-
A () A () Bester () A () Bester ()	all - (2 - 70 weeks stepping own and Atennes Seson 2 - 3 - Chapter Struct, ample Scapelist app al Search trace Schemiss Tells (2 min ex-Sundry craditors (2 types).	neute latonaet beponen neute indox. 21 Does Media: @Windows	-
27 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	eur - ((? - Totokonas	newski leitonaet beginnen nextis visiot. 3 3 ons %ede	
States (try footoof/coloring) the Commentant (from the coloring) the Coloring (Contallist) the coloring (Contallist) the coloring (Contallist)	ett - ((2 - 1) Steenens	neuto tetoret lagorer neuto tetor. 3 Jose mate @lemans	
A A	ett - ((?-))downess _stempleptoon een @revets @reson []- @ (Depresting)_empediates # @revet toore stensonstes @reve ex-Suntry crattors @typest.	Internet Laborate Explorer Institution	
And the sees	ett - ((? - 1) Svenenis _sterpting to on exec _() reverses	Elle Current Sabifies	
Annual Control Control Control Annual Control Control Annual Contro Annual Control Annual Control Annual Control	If "Otherman Interplay Com- Comparison (Comparison Comparison) Comparison (Comparison) Comparison (Comparison) Comparison (Comparison) Comparison Comparison	Elle Current Sabifies Elle Current Sabifies Elle Current Sabifies Elle Current Sabifies	
A A A A A A A A A A A A A A A A A	att - ((1 - 1) Obviouslystempling to compare the secondstempling to compare the second compare the secon	Edit Current Sabirities Edit Current Sabirities	
Antipartition of the second and a second and a second and a second at second base	If "Industrial Interplay to on Arrows General Constants Constanting Comparison Constants Constanting Comparison Constanting Comparison Constanting Comparison Constanting Constants Constanting Constants Constanting	Elle Current Sabities Elle Current Sabities Elle Current Sabities Elle Current Sabities Elle Deserves and surplas Elle Deserves and surplas	
And the second s	It 'n Devices's steeping to on Arrowski Greater (Greater Councils and Contraction of the second of th	Edit Durier Explorer Insuffer Televite Edit Current Sabirities Edit Current Sabirities Edit Current Sabirities Edit Current exets Edit Current exets Edit Duties taxes peradic (ba) Edit Back of & limits.	
A Capital Second S	It - Trobusenia stempting to on Comparison (Comparison (Comparison)) Comparison (Comparison) Comparison (Comparison) Comparison (Comparison) Comparison Compa	Else Current Sabinies	ne "
Arrow and a second data a	If "Industrial stepping to on and (If "Industrial Second	Edit Durinet Explorer Institutional Edit Varianti Edit Current Explorer Edit Current Explorer Edit Current Explorer Edit Current Explorer Edit Duties taxes perable (be) Edit Duties taxes perable (be) Edit Duties taxes perable (be)	
And A second	It "Industrial atompting to one Arrendom to Sensor (a) revolute (Sensor) (a) (a) (b) Construction production again Sensor to one submession bases Sensor to one submession babaaaa	Edd During Explored Insuff United.	
Arrow and a second and a second a second a second a second at the s	If "Indexemia stempting to one Origination (Second Second Secon	Edd Current Relations Edd Current Relations Edd Current Relations Edd Current Relations Edd Current Sets Edd Current Assets Edd Current Asse	780 ¹⁰
Provision Control of the second	If "Industrial strapping to on and If "Industrial Second I - Contraction of the second I - Contrac	Elle Durine Explorer Elle Variout. 2 2 2 2 2 2 2 2 2 2 2 2 2	

Figure 4.7: DataList in Edit Mode. GroupsDlist.aspx

<%@Page Language="VB" Inherits="BaseClass" Src="GroupsDlist.vb" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server">

Sub DataList_UpdateCommand(sender As Object, e As DataListCommandEventArgs)

Dim sql As string

Dim code_display As String

Dim type As String

Dim myTextBox As TextBox

myTextBox = E.Item.FindControl("edit_display")

code_display = mytextbox.text

myTextBox = E.Item.FindControl("edit_type")

type = mytextbox.text

'Now execute stored procedure

response.write("Execute some procedure @name=" + code_display + "@type=" + type)

End Sub

</script>

<body style="font: 10pt verdana">

<form runat="server">

<h3>Groups (DataList) </h3>

<asp:Label id="Message" runat="server"/>

<asp:DataList id="Grid1" runat="server"

BorderColor="black"

BorderWidth="1"

GridLines="Both"

CellPadding="3"

CellSpacing="0"

Font-Name="Verdana"

Font-Size="8pt"

Width="800px"

HeaderStyle-BackColor="#aaaadd"

AlternatingItemStyle-BackColor="Gainsboro"

EditItemStyle-BackColor="lightgreen"

OnEditCommand="DataList_EditCommand"

OnUpdateCommand="DataList_UpdateCommand"

OnCancelCommand="DataList_CancelCommand"

RepeatColumns="3" RepeatDirection="horizontal" RepeatMode="Table" >

<HeaderTemplate>Name</HeaderTemplate>

<ItemTemplate>

<asp:LinkButton id="button1" runat="server" Text="Edit" CommandName="edit" />

<%# Container.DataItem("code_display") %>

</ItemTemplate>

<EditItemTemplate>

Name:

<asp:Label id="Label1" runat="server" Text='<%# Container.DataItem("code_display") %>' />

Group:

<asp:TextBox id="edit_display" runat="server" Text='<%# DataBinder.Eval(Container.DataItem,

```
"code_display") %>'/>
```


Type:

<asp:TextBox id="edit_type" runat="server" Text='<%#
DataBinder.Eval(Container.DataItem, "type") %>' />

<asp:LinkButton id="button2" runat="server" Text="Update" CommandName="update" />

<asp:LinkButton id="button3" runat="server" Text="Cancel" CommandName="cancel" />

</EditItemTemplate>

</asp:DataList>

</form>

</body>

</html>

GroupsDlist.vb is the Code Behind file for this form. It contains the following code: GroupsDlist.vb

Option Strict Off Imports System Imports System.Collections Imports System.Text Imports System.Data Imports System.Data.OleDb Imports System.Web.UI Imports System.Web.UI.WebControls Public Class BaseClass Inherits System.Web.UI.Page Protected Grid1 As Datalist Protected Message as label Dim myConnection As OleDbConnection Dim myCommand As OleDbDataAdapter Dim ds As New DataSet Dim ConnStr As String Dim SQL As String Sub Page_Load(Source As Object, E As EventArgs) ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;" myConnection = New OleDbConnection(ConnStr) if NOT (isPostBack) rebind end if

End Sub

Sub ReBind()

'DataSetCommand

SQL = "select * from Groups"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "Groups")

'Binding a Grid

Grid1.DataSource=ds.Tables("Groups").DefaultView

Grid1.DataBind()

End Sub

Sub DataList_EditCommand(sender As Object, e As DataListCommandEventArgs)

Grid1.EditItemIndex = e.Item.ItemIndex

rebind

End Sub

Sub DataList_CancelCommand(sender As Object, e As DataListCommandEventArgs)

Grid1.EditItemIndex = -1

rebind

End Sub

End Class

The DataList derives its data from the DataSet ds. The function Rebind populates the DataList with data. In the ItemTemplate I define a template for LinkButton which displays a link called "edit". An "Edit" CommandName has been defined on the LinkButton. When clicked the CommandName tells the DataList that it is in Edit mode. Once in Edit mode, the DataList automatically generates the "update" and "cancel" links. Clicking on the "edit," "update," or "cancel" links fires the "DataList_EditCommand", "DataList_UpdateCommand", or "DataList_CancelCommand" respectively. The EditItemIndex property of the DataList is set to the index of the clicked button in the DataList_EditCommand event. The EditItemIndex is set to -1 in the cancel event. In the update command (DataList_UpdateCommand) I build a SQL string in a manner similar to the one discussed in the DataColumns="3" and RepeatDirection= "horizontal" properties in the DataList tag. This generates 3 columns across in the horizontal direction.

I use the DataBinder.Eval() method to bind the columns. This method takes three arguments: the naming container for the data item, the data field name, and a format string. The datacontainer for a DataList (and DataGrid) is always Container.DataItem. You can apply formatting to the bound column such as in the following:

<%# DataBinder.Eval(Container.DataItem, "IntegerValue", "{0:c}") %>

Binding to XML Data

XML is an integral part of ASP.NET so it is very simple to bind a list control to an XML datasource. Figure 4.8 shows an example.

Reading X	ML					
id	uri	headine_text	pource	media_type	cluster its	gine document_url
_11052107	http://c.moreower.com/click/here.pl? x11052105	Heavy raise flood routhern Brasil	MSNEC	teat	Branl arws	htp://www.mashe.com
_11050007	http://c.moreover.com/clicichere.pi7 x11050005	Heavy rains kill six in routhern Brazil	AP via New Jerrey Online	reat	Brazil arws	http://www.nj.com/nes iztemational.html
11035043	http://c.moreover.com/click/here.p/? x11035032	Brazil shares end at 4-1/2 month low on oil jitters	Individual com via Oil Neuro	best	Brazil news	htp://www.oilnews.co

Figure 4.8: Binding to XML Data. navXML.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.IO" %>

<html>

<head> Reading XML </head>

<script language="VB" runat="server">

Sub Page_Load(Source As Object, E As EventArgs)

Dim ds As New DataSet

Dim fs As filestream

Dim xmLStream As StreamReader

fs = New filestream(Server.MapPath("more.xml"), FileMode.Open, FileAccess.Read)

xmlStream = new StreamReader(fs)

ds.ReadXML(XmlStream)

DataGrid1.DataSource=ds.Tables("article").DefaultView

| DataGrid1.DataBind() |
|---|
| End Sub |
| |
| <body></body> |
| <form runat="server"></form> |
| <asp:datagrid id="DataGrid1" runat="server"></asp:datagrid> |
| |
| |
| |
| |

I use the filestream method to open the XML file (more.xml) in a "read" mode and populate the StreamReader with the contents. I then use the ReadXML method of the DataSet to read the XML data into the DataSet. Once in the DataSet, I can manipulate the data in a manner similar to database tables. Finally, I bind the "article" element of the XML file to a DataGrid.

Implementing a Master-Detail Relationship

A master-detail type of relationship is when one to many relationships exist between two tables. The master table rows are shown in one portion of the form. Clicking on a master row displays all detail rows in another location of the form. I will show you how to implement a master-child relationship using the authors, titles, and titleauthor tables of the Pubs database. The Authors details are displayed with a "Select" link. Clicking on this hyperlink displays details about the books written by the selected author. This relationship is implemented using a DataGrid. Figure 4.9 illustrates what it looks like.

| dorma (e
nic a): | • (E) http://oc | Contraction of the second seco | Favori
Favori
(chap) or
Search Eng | es S History
Inple: MacerChic
pre Submission Sh | C-@3
Lacx
e: 2)Windows Neda 2)W | indows | |
|---------------------|-----------------|--|---|--|---------------------------------------|-----------|----------|
| Mas | ster | Child Exa | mpl | e
Titles: | | | |
| | 10 | Name | State | 10 | Title | Published | Price |
| Select | 172-92-
1176 | White Johnson | CA | | Cooking with
Computers | 100 1991 | 671.95 |
| Salect | 213-46-
0915 | Green,Marjorie | CA | | Surreptitious balance
Sheets | 201 2192 | \$1.1.75 |
| Select 2 | 213-46- | Green,Marjorie | CA | TC7777 | Dushi, Assere? | Jun 1991 | \$14.33 |
| inker | 238-95-
7766 | Carson, Cheryl | CA | | | | |
| Salect | 267-41- | O'Leary,Michael | ¢A | | | | |
| Select | 267-41-2394 | 0'Leary,Nichael | C.A. | | | | |
| Salect | 274-80+
9391 | Straight, Dean | CA | | | | |
| Sales 1 | 409-58-7000 | besset,Abraham | CA | | | | |
| Salest 3 | 427-17+ | Dull,Ann | CA | | | | |
| Sales! | 472-27- | Gringlesby,Burt | CA | | | | |
| | and no | | | | | | |

Figure 4.9: Master Child Relationship. MasterChild.aspx

<%@Page Language="VB" Inherits="BaseClass" Src="MasterChild.vb" %>

<html>

<head>

<title>DataGrid Samples - Step 3</title>

</head>

<body>

<h1> Master Child Example </h1>

<form runat=server>

Authors:

<asp:DataGrid id="authorsGrid" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

DataKeyField="au_id"

OnSelectedIndexChanged="Grid_Select"

>

<Columns>

<asp:ButtonColumn Text="Select" CommandName="Select"/>

<asp:BoundColumn HeaderText="ID" DataField="au_id">

<HeaderStyle Width="100px"> </HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Name" DataField="au_name">

<HeaderStyle Width="150px"></HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="State" DataField="state">

<HeaderStyle Width="50px"></HeaderStyle>

</asp:BoundColumn>

</Columns>

<HeaderStyle BackColor="DarkRed" ForeColor="White" Font-Bold="true">

</HeaderStyle>

<ItemStyle ForeColor="DarkSlateBlue">

</ltemStyle>

<AlternatingItemStyle BackColor="Beige">

</AlternatingItemStyle>

<SelectedItemStyle BackColor="PaleGoldenRod" Font-Bold="true">

</SelectedItemStyle>

</asp:DataGrid>

<asp:Panel id="detailsPanel" runat="server" Visible="false">

Titles:

<asp:DataGrid id="titlesGrid" runat="server"

AutoGenerateColumns="false"

ShowFooter="true"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan" CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

>

<Columns>

<asp:BoundColumn HeaderText="ID" DataField="title_id">

<HeaderStyle Width="100px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Title" DataField="title">

<HeaderStyle Width="250px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Published" DataField="pubdate"

DataFormatString="{0:MMM yyyy}">

<HeaderStyle Width="100px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Price" DataField="price" DataFormatString="{0:c}">

<HeaderStyle Width="50px">

</HeaderStyle>

<ItemStyle HorizontalAlign="Right">

</ltemStyle>

</asp:BoundColumn>

</Columns>

<HeaderStyle BackColor="DarkRed" ForeColor="White" Font-Bold="true">

</HeaderStyle>

<FooterStyle BackColor="Tan">

</FooterStyle>

<ItemStyle ForeColor="DarkSlateBlue">

</ltemStyle>

<AlternatingItemStyle BackColor="Beige">

</AlternatingltemStyle>

</asp:DataGrid>

</asp:Panel>

</form>

</body>

</html>

The Code Behind for this form is MasterChild.vb: MasterChild.vb

Imports System

Imports System.Collections

Imports System.Text

Imports System.Data

Imports System.Data.OleDb

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class BaseClass

Inherits System.Web.UI.Page

Protected AuthorsGrid as DataGrid

Protected titlesGrid as DataGrid

Protected detailsPanel as Panel

Public currentAuthor as object

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

```
ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=pubs;User
ID=sa;"
  myConnection = New OleDbConnection(ConnStr)
  if NOT (isPostBack)
   rebind
  end if
 End Sub
 Sub FillDs
  sql = " select * , au_lname + ',' + au_fname as au_name"
  sql = sql + " From authors a, titles t, titleauthor ta"
  sql = sql + " Where a.au_id = ta.au_id AND t.title_id = ta.title_id"
  myCommand = New OleDbDataAdapter(SQL, myConnection)
  'use Fill method of DataSetCommand to populate dataset
   myCommand.Fill(ds, "Authors")
 End Sub
 Sub ReBind()
  FillDs
  'Binding a Grid
  AuthorsGrid.DataSource=ds.Tables("Authors").DefaultView
  AuthorsGrid.DataBind()
 End Sub
 Sub Grid_Select(sender as Object, e as EventArgs)
  Dim vIndex As Integer
  Dim vkey As string
  vIndex = AuthorsGrid.SelectedIndex
  vkey =AuthorsGrid.DataKeys(vIndex).ToString
  UpdateSelection(vkey)
 End Sub
 Sub UpdateSelection(vkey as string)
```

Dim myConnection2 As OleDbConnection

Dim myCommand2 As OleDbDataAdapter Dim ds2 As New DataSet Dim ConnStr2 As String **Dim SQL2 As String** Dim itemcount As Integer sql2 = " select * , au_Iname + ',' + au_fname as au_name" sql2 = sql2 + " From authors a, titles t, titleauthor ta" sql2 = sql2 + " Where a.au_id = ta.au_id AND t.title_id = ta.title_id" sql2 = sql2 + " AND a.au_id = '" + vkey + "'" myCommand2 = New OleDbDataAdapter(SQL2, myConnection) myCommand2.Fill(ds2, "Authors") 'Bind the Grid titlesGrid.DataSource=ds2.Tables("Authors").DefaultView titlesGrid.DataBind() itemcount = titlesGrid.Items.Count if itemcount >= 1 then detailsPanel.Visible = true Else detailsPanel.Visible = false response.write("No rows found") end if

End Sub

End Class

The Master-Grid implementation is comprised of two DataGrids: the AuthorsGrid and the titlesGrid. The AuthorsGrid is bound to a query, which is a join between the Authors, titles, and TitleAuthors table. The query is as follows:

sql = " select * , au_lname + ',' + au_fname as au_name"

sql = sql + " From authors a, titles t, titleauthor ta"

sql = sql + " Where a.au_id = ta.au_id AND t.title_id = ta.title_id"

I have created a ButtonColumn with CommandName = "select". The "select" CommandName informs the DataGrid that an item has been selected and fires the "OnSelectedIndexChanged" event, which in turn fires the "Grid_Select" function. The Grid_Select function retrieves the au_id (the DataKeyField) of the row that has changed and calls the UpdateSelection function. The UpdateSelection function binds the titlesGrid with the following query:

sql2 = " select * , au_lname + ',' + au_fname as au_name"

sql2 = sql2 + " From authors a, titles t, titleauthor ta"

sql2 = sql2 + " Where a.au_id = ta.au_id AND t.title_id = ta.title_id"

sql2 = sql2 + " AND a.au_id = '" + vkey + "'"

This query limits the original query to the selected author only. Note that the titlesGrid is enclosed within a Panel which has an id of DetailsPanel, which is initially invisible. The UpdateSelection binds the titlesGrid and makes the Panel visible. This in turn displays the detail records.

Summary

I have covered quite a lot of ground in this chapter. I have shown how the "list bound" controls are bound to a datasource. Practical examples of using the DataRepeater, the DataList, and DataGrid were provided. Working with XML datasources was also explained. Further chapters in this book build on these concepts and the project section of the book shows how to use them in actual applications.

Chapter 5: Input Validation

Overview

Input validation is a dull, dreary task. I don't see too many developers getting excited about writing validation code. However, the value of good validation techniques cannot be discounted. Without appropriate validation, our script routines would break, and we would get garbage in our database. ASP.NET makes the task of implementing validation routines a breeze. Implementing a validation routine is as simple as creating a validation control and telling it what control to validate.

The ASP.NET team did detailed research of a number of data entry forms. They found that most validation tasks revolved around the following activities:

- Checking for "regular expressions" such as ZIP codes and telephone numbers
- Comparing two users' input values
- Checking for required fields
- Checking if a given input falls within a range of values

In addition they found that the user could be informed of a wrong input value immediately. Also, all wrong input values could be summarized and shown together. Armed with this knowledge they set out to develop an object that could encompass all these activities and take the yoke of writing validation code off our necks. Writing such an object in an ActiveX environment would have meant overloading the functionality of all the requirements listed above into a single component that behaved differently in different modes. However, the .NET framework allowed them to create six controls that all inherit from a common object (BaseValidator). Each object specializes in providing certain functionality. Since the scope of each object is focused, it is a lean, but very effective control. The following list shows the controls they came up with:

RequiredFieldValidator

- RegularExpressionValidator
- CompareValidator
- RangeValidator
- CustomValidator
- ValidationSummary

There are five validation controls and one ValidationSummary control. The work of the first four validation controls is evident from their names. You can write your own validation function and associate it with the CustomValidator. Finally, the ValidationSummary control presents a summary of all the errors on the page in one location.

A Two-Pronged Approach to Validation

A good validation approach requires that you validate user input both at the client and server side. Client side validation is nice to have. Users get immediate feedback of illegal input values. The downside of client side validation is that it is quite cumbersome to implement using just HTML 3.2. Scripting languages and DHTML make this task easy. However, this makes an assumption that users will use browsers that support these technologies. This is not a valid assumption to make. Using only client side validation can also pose a security risk. It is quite easy to tamper, replace, or bypass a script page. It is for this reason that the validation controls use a two-pronged approach. They use the client side validation to provide immediate feedback and then repeat the validation at server side. Client side validation is automatic with Internet Explorer 4.0 and above. For script-disabled browsers, validation is carried out server side.

The client side validation has number of features. An immediate feedback (say an error message in red) is given to users regarding illegal entries, which goes away after the error is rectified. If an error is trapped at the client side, a post back to the server is avoided. The validation summary updates itself, again without a post back. No ActiveX objects or applets are used to implement the client side functionality as the logic is contained in a JScript library.

Validation Controls

I have created a Web page with a sample of each validation control. I shall be using this in my discussions. <u>Figure 5.1</u> displays the result of the validate.aspx code.

| 🗿 Validating with ASP + Controls - Microsoft Internet Explorer | |
|---|---------------------------|
| " 27 1 ? 1 Search - Ouert Isaker Sgn n @ W Valcor Pews -
○ 2 @ @Search 1 Fearches @Hetay [2 - @ 2] | 30 MB * |
| Address & http://incahod.jAphietBook/ChepterS/Serpleshakdete.espo | • ര'ഞ |
| unie @]fop-Computers-Programming @]Costonize unie @]Pree Hornal @]Search Englie Subression Stes | @]Windows Media @]Windows |
| Required Field : | - |
| Name : | |
| Password: | |
| Range Validator | |
| IQ (180 - 255):
Rogular Expression: | |
| validation expression : [0-9]{3}/\s[0-9]{3}-[0-0]{4}
Example : 214 345-0458 | |
| Phone. | |
| Custom Validation: | |
| Must be the number 10 | |
| Submit | |
| e) love | ELicalistanet |



<%@ Page language="VB" %>

<html>

<head>

<script runat=server>

Sub ServerValidate (sender As Object, value As ServerValidateEventArgs)

```
Dim num As Int32 = Int32.Parse(value.Value)
```

response.write(num)

if num= "10" then

value.IsValid = True

else

value.IsValid = False

end if

End Sub

public sub OnSubmit(source as Object, e as EventArgs)

if Page.IsValid then

```
' Check before update to database
```

end if

end sub

</script>

```
<title>Validating with ASP+ Controls</title>
```

</head>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

```
<h2>ASP .NET Control Validation</h2>
```

<hr>

```
<asp:ValidationSummary runat=server headertext="There were errors on the page:" />
```

<form RunAt="server">

Required Field :

Name : <input type=text runat=server id=txtName>

<asp:RequiredFieldValidator runat=server

controltovalidate=txtName

errormessage="Name is required.">*

</asp:RequiredFieldValidator>

<hr>

Password:<asp:TextBox id="txtPassword" RunAt="server"/>

Confirm:<asp:TextBox id="txtConfirm" RunAt="server"/>

<asp:CompareValidator

id="cvPassword"

ControlToValidate="txtPassword"

ControlToCompare="txtConfirm"

Type="String"

Operator=Equal

Display="dynamic"

ErrorMessage="The password does not match the confirm password!"

RunAt="server">

</asp:CompareValidator>

<hr>

Range Validator

IQ (180 - 265):<asp:TextBox id="iq" RunAt="server"/>

<asp:RangeValidator

id="rvIQ"

ControlToValidate="iq"

Display="dynamic"

ErrorMessage="IQ must be between 180 and 265 to run this example"

MinimumValue=180

MaximumValue=265

Type="Integer"

RunAt="server">

</asp:RangeValidator>

Regular Expression:

validation expression : [0-9]{3}\s[0-9]{3}-[0-9]{4}

Example : 214 345-0458

> Phone:<asp:TextBox id="txtPhone" RunAt="server" /> <asp:RegularExpressionValidator id="revPassword" ControlToValidate="txtPhone" Display="dynamic" ValidationExpression="[0-9]{3}\s[0-9]{3}-[0-9]{4}" ErrorMessage="Phone number must be xxx xxx-xxxx" RunAt="server"> </asp:RegularExpressionValidator> <hr> Custom Validation:

 Must be the number 10 <asp:TextBox id=Text11 runat="server" /> <asp:CustomValidator id="CustomValidator1" runat="server" ControlToValidate="Text11" OnServerValidate="ServerValidate" Display="Static" Font-Name="verdana" Font-Size="10pt"> This field must be the number 10! </asp:CustomValidator>
 <hr> <input type=submit runat=server id=cmdSubmit value=Submit onserverclick=OnSubmit>

</form>

</body>

</html>

Required Field

The required field validation control ensures that specified fields get filled in. The example of a required field is as follows:

Name : <input type=text runat=server id=txtName>

<asp:RequiredFieldValidator runat=server

controltovalidate=txtName

errormessage="Name is required.">*

</asp:RequiredFieldValidator>

The user is supposed to fill out the txtName textbox. If he leaves it blank and hits the submit button, an error is displayed ("Name is required") by the ValidationSummary control. An asterisk (*) is also displayed next to the textbox. The ControlToValidate property specifies the ID of the control to validate, which in this case is txtName.

Compare Validator

The CompareValidator compares the content of two controls and if they don't match, reports an error. A typical use of the CompareValidator is to match a password and a password reentry value. The following is an example:

Password:<asp:TextBox id="txtPassword" RunAt="server"/>

Confirm:<asp:TextBox id="txtConfirm" RunAt="server"/>

<asp:CompareValidator

id="cvPassword"

ControlToValidate="txtPassword"

ControlToCompare="txtConfirm"

Type="String"

Operator=Equal

Display="dynamic"

ErrorMessage="The password does not match the confirm password!"

RunAt="server">

*

</asp:CompareValidator>

Here I am comparing two fields. If they don't match, a * is displayed and the error message "The password does not match the confirm password!" is displayed by the validation control when the submit button is clicked. You can specify the data type of the values and the comparison operator.

- If you get rid of the * in the CompareValidator tag, the error message is displayed instead of the * to provide immediate feedback to the user.
- 2. The Type property can be of the following:
 - String
 - Integer
 - Double
 - DateTime
 - Currency
- 3. The Operator property can be of the following:

- Equal
- NotEqual
- GreaterThan
- GreaterThanEqual
- LessThan
- LessThanEqual
- DataTypeCheck
- 4. The Display property can be of the following:
 - None. Indicates that no inline error message should be displayed. However, the control will still get evaluated and the validation control will display the error as a summary.
 - Dynamic. Indicates that the control will take page space when the error text is displayed resulting in page layout change. This is useful when multiple validation controls are attached to a single control.
 - Static. Indicates that space will be reserved for the full error message. In this case the page layout will not change when the error message is displayed.
- 5. A CompareValidator is valid if blank.

Range Validator

You use the RangeValidator control to check if the user input falls within a specified range. In the example that follows, the RangeValidator checks if the input IQ falls within 180–265.

Range Validator

IQ (180 - 265):<asp:TextBox id="iq" RunAt="server"/>

<asp:RangeValidator

id="rvIQ"

ControlToValidate="iq"

Display="dynamic"

ErrorMessage="IQ must be between 180 and 265 to run this example"

MinimumValue=180

MaximumValue=265

Type="Integer"

RunAt="server">

</asp:RangeValidator>

The RangeValidator has the ControlToValidate, Display, and Type properties, which have already been discussed. In addition it has a MinimumValue and a MaximumValue property. These are the upper and lower bounds of the range within which a user input must fall. The RangeValidator is valid if blank.

Regular Expression

The RegularExpressionValidator is the most powerful of all the controls. It checks for user input against a pattern of characters specified by the developer. Regular Expressions have syntax of their own. For example, to ensure that a telephone number is entered in the format "xxx xxx-xxxx," you could enter this regular expression "[0-9]{3}\s[0-9]{3}-[0-9]{4}". This means that the first three characters should fall within 0 to 9, followed by a blank space (\s), followed by three characters which also fall between 0 to 9, followed by a dash, and then four characters which fall within 0 to 9. You can use this control to check valid e-mail identifications, ZIP codes, and in fact any user-defined pattern. I shall be looking at the RegularExpressionValidator, which checks that a telephone number is of the pattern "xxx xxx-xxxx."

Phone: <asp:TextBox id="txtPhone" runat="server" />

<asp:RegularExpressionValidator

id="revPassword"

ControlToValidate="txtPhone"

Display="dynamic"

ValidationExpression="[0-9]{3}\s[0-9]{3}-[0-9]{4}"

ErrorMessage="Phone number must be xxx xxx-xxxx"

runat="server">

</asp:RegularExpressionValidator>

You will note that the RegularExpressionValidator has the ControlToValidate, Display, and ErrorMessage properties. These properties have the same connotation as the other validation controls. The Validation property is where you set the regular expression pattern. The RegularExpressionValidator is valid if blank.

A Regular Expression Pattern Primer

A regular expression is a sequence of characters that provides a pattern (or template) against which to match a value input by the user.

Pattern Matching

The pattern "Hersh" matches string combinations of characters "Hersh" appearing together and in that order. "Her.h" would match "Herjh," "Hersh," and "Herdh." The decimal point matches any character except the newline character.

Now suppose you wanted to precede "Hersh" with a digit falling within the range 0–9, you would say "[0-9]Hersh." The "[xyz]" brackets represent a character set and any of the enclosed characters are matched. The pattern "[a-z]" represents lower case alphabets whereas "[a-zA-Z]" includes all uppercase and lowercase alphabets. To search for digits in a string, use "[0-9]."

Repetition Matching

Suppose you wanted to find all occurrences of "book" and "books." You would use the pattern "book?." The question mark matches the preceding character 0 or 1 times (i.e. it will only look at one more character after "book" and concatenate it to "book"). Now suppose we wanted to go beyond the one character limit imposed by the question mark and search for "book" followed by anything. We would use the "*", and our pattern will look like "book*." This will return words like "booked," "bookiee," "bookworm," etc.

The expression "{n}" matches against the target string exactly n times. For example, "p{2}" will match "happy" but not "hop."

Position Matching

Suppose you wanted to find words which started with "Gr," you would use the caret symbol and your pattern would be "^Gr." This would match words like "Great," "Grate," "Grr," etc. The caret symbol matches the start of the line.

The dollar symbol (\$), on the other hand, matches the end of the line. Thus if you wanted to find words ending with "d" your pattern would be "d\$" and this would fetch words like "had," "bad," "lad," etc.

Custom Validation

The CustomValidator can be used to extend the functionality of the validation controls by performing validation using custom functions and routines. This is particularly useful in cases when you need to access database information as part of the validation routines. You can define a server function and optionally a client side function that gets called by the CustomValidator. The functions on both client and server side would be similar. There is no need of having elaborate client side routines, as the client side checks can easily be side-stepped by a malicious user. You should only use client side routines to give immediate feedback to users when an illegal entry is made. The real validation functionality should be maintained server side.

The following is an example:

Custom Validation:

Must be the number 10

<asp:TextBox id=Text11 runat="server" />

<asp:CustomValidator id="CustomValidator1" runat="server"

ControlToValidate="Text11"

OnServerValidate="ServerValidate"

Display="Static"

Font-Name="verdana" Font-Size="10pt">

This field must be the number 10!

</asp:CustomValidator>

In this example the user is supposed to enter the number 10. Input of any other number causes an error state and the client side custom error function gets fired. When the submit button is pressed, the server side error function gets fired.

Server Side Functionality

The server side function is specified by the <code>OnServerValidate</code> property. This has been set to fire the function <code>ServerValidate</code>, which is as follows:

Sub ServerValidate (sender As Object, value As ServerValidateEventArgs)

Dim num As Int32 = Int32.Parse(value.Value)

```
response.write(num)
```

if num= "10" then

value.lsValid = True

else

value.lsValid = False

end if

End Sub

This simply checks if the number entered is 10. If not, it returns false.

Client Side Functionality

The client side functionality is maintained by the property ClientValidationFunction. This has been set to fire the custom function ClientValidate. This can be a function written in JScript or VBScript. I show a skeleton function in the following code:

<script language = javascript >

<!--

```
function ClientValidate(source,value) {
```

```
// Put some client side code here
```

} //-->

</script>

The code in this function gets fired each time the user enters something in the field and tabs out. It will return true or false, which will in turn toggle the error state on or off. For older browsers or when the client validation has been turned off, this function will not get called. The CustomValidation control automatically checks the browser capabilities and renders HTML accordingly. However, you should always enclose the client side script in HTML comments so that older browsers ignore it. Two parameters are passed into the client function (which correspond to parameters passed to the server function). These are the client validator element and the value of the control specified by the ControlToValidate. In the client side function you can choose to ignore these parameter definitions in your function call. Finally, the CustomValidator is valid if blank.

The ValidationSummary Control

The final control is the ValidationSummary control. This control accumulates values from the errormessage property of all the validation controls on the page and displays them together. You can place the validation control anywhere on the page. You can also control how the errors are displayed (bulleted, plain list, etc). The following is an example:

<asp:ValidationSummary runat=server

displaymode ="bulletlist"

showsummary = "true"

headertext="There were errors on the page:" />

The validation control has the following properties:

- HeaderText. This is the caption of the errors list.
- DisplayMode. This can be List (separated by
), BulletList (defaultseparated by), and SingleParagraph (un-delimited, all in one para).
- ShowSummary. True (default) or False. If set to false, only HeaderText is displayed.
- ShowMessageBox. True or False (default). When set to true, shows a pop-up message box instead of the summary.

The IsValid Property

The IsValid property is a page property, which evaluates to true if no errors were encountered or false if errors were found. It is important to check this property before making updates, etc., to the database. The following is how a submit handler might look:

Public sub OnSubmit(source as object, e as eventargs)

If Page.IsValid then

'proceed to update database

End if

End Sub

Disabling Client Side Validation

At times you may want to disable client side validation. You might have code that needs to be executed server side only and client side validation might be unnecessary, or you might have too few input fields to justify client side validation. To disable client side validation you use a Page Directive "clienttarget = downlevel". It looks like the following:

<%@ Page Language = "vb" clienttarget = "downlevel" %>

The default value for this directive is "auto" meaning that you get client side validation for Internet Explorer 4.0 and above only.

Summary

The validation controls take out the drudgery of writing validation code. They are intelligent beasts and can render HTML based on the browser type. They are a welcome addition to our toolbox and I will be making extensive use of them in later chapters.

Chapter 6: User Controls

"Code Behind" techniques involved separating the visual portion and the scripting portion into separate forms. The web form basically inherited the code classes into the page. A *user control*, however, can be a self-contained entity, comprised of both the graphical interface and scripting code. A user control has the functionality (and simplicity) of legacy ASP Server Side Includes. However, include files are static files whereas user controls provide an object model support, which allows you to program against properties and use methods. They work much like the ASP.NET intrinsic controls, and like intrinsic controls they can expose properties and methods. You need to choose a single language when writing ASP.NET web forms. User controls circumvent this restriction as you can have controls written in different languages on the same Web page. They can also be used more than once on the same page without having any naming conflicts. This is because each control resides on its own namespace.

A user control file is a simple text file, saved with the extension ascx. It should not contain the <html> and <body> and the <form> tags. The page that calls the user object will apply these tags. Any existing web form can be converted to a user control with slight modifications.

Creating a Simple User Control

I will now create a user control, which displays the single line of text, "This is a Test Control," when invoked. See <u>Figure 6.1</u> for an example.



Figure 6.1: Simple user control.

1. Create a file and save it with the name simpleUC1.ascx.

2. Add a single line of text in this file "This is a Test Control."

3. Create a web form simpleUC1.aspx and add the following tags:

simpleUC1.aspx

<%@ Register TagPrefix="Hersh" TagName="test" Src="SimpleUC1.ascx" %>

<html>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<Hersh:test runat = server />

</body>

</html>

4. Run the web form in your browser.

The register directive registers the control with the Web page. I have given the user control a TagPrefix of "Hersh". This can be any name. The TagName is the alias to the user control. We use this to refer to the control. You can see in this simplified example that the user control behaves much like an include file.

Exposing Properties

Visual Basic.NET and C# provide a useful way of assigning and retrieving values through accessor and mutator functions. Instead of the Let, Set, and Get statements in VB 6.0 (which are now unsupported), you just need a single Property block that would contain a Set and Get block. In this example I have a "message" property block as follows:

Public Property message As String

Get

Return Txt.Value

End Get

Set

Txt.Value = Value

End Set

End Property

The accessor functions (the Get functions) get called when values are requested from the message property. Similarly, the mutator function (the Set function) gets called when values are assigned to the message property.

A user control can expose properties, which can then be set by the user. The contents of the user control can thus be encapsulated so that the user is not exposed to the inner working of the component. In the following example, my user control is comprised of a single textbox to which I pass a string "Message passed from web form." Figure 6.2 shows what the output of the web form looks like.



Figure 6.2: Exposing properties in user controls. simpleUC2.aspx

<%@ Register TagPrefix="Hersh" TagName="test" Src="SimpleUC2.ascx" %>

<html>

```
<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">
```

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

```
<Hersh:test runat = server message="Message passed from web form" />
```

</form>

</body>

</html>

The user control form contains the following code: **simpleUC2.ascx**

<script language="VB" runat="server">

```
Public Property message As String
```

Get

Return Txt.Value

End Get

Set

Txt.Value = Value

End Set

End Property

</script>

<input id="Txt" size="50" type="text" runat="server">

User controls reside on their own namespace. This means that the variable names specified for a user control will not clash with another user control, intrinsic control, or even the page variables.

It is recommended that you expose properties instead of Public variables. Properties allow data hiding, can be versioned, and are supported by visual designers such as Visual Studio.NET. I have defined a property called message, which assigns the passed message text to the textbox.

Designing a Navigation System for Your Web Site with a User Control

Each time a new Web page is added to your site, you need to update all the web forms in the site to update the navigation structure. Site navigation should be kept separate from your web forms. When I build a site, I keep all my links in an XML file. I then merge my "navigation" XML file with my "main" XML file and render them by applying stylesheets. I used to do this using a combination of MSXML and XSL stylesheets and was pleasantly surprised to find that implementing the same functionality in ASP.NET was a breeze.

I maintain my site links in an XML file with the following structure: nav.XML

<Siteinfo>

<site>

<sitename>Home</sitename>

<siteurl>default.aspx </siteurl>

</site>

<site>

<sitename>Masters</sitename>

<siteurl>masters.aspx</siteurl>

</site>

</Siteinfo>
The user control that I developed can be placed on any web form with an invocation like the following:

Navigation.aspx

<%@ Register TagPrefix="Hersh" TagName="nav" Src="nav.ascx" %>

<html>

<head>

<style>

a { color:black;

text-decoration:none;}

a:hover { color:red;

text-decoration:underline;}

</style>

</head>

<body style="background-color='white'; font-family='verdana'; font-size='10pt'">

<form runat=server>

<Hersh:nav id="menu" runat = server

vGridlines = Both

vBorderColor = "Black"

vCellPadding = 7

/>

</form>

</body>

</html>

This displays a navigation menu, which looks like Figure 6.3.

E)top-C	orguters - Programming @] (Cusionize Links @Pree Hornel @35	serb Brighe Subvession Stes 국가에게dows Media 국가에
ne	<u>Hartora</u>	Transections	Trial Belance

Figure 6.3: Navigation Menu.

The user control is contained in the file nav.ascx. A Web page that calls the control will register it with a directive like the following:

<%@ Register TagPrefix="Hersh" TagName="nav" Src="nav.ascx" %>

The control is then instantiated with the following tags:

<form runat=server>

<Hersh:nav id="menu" runat = server

```
vGridlines = Both
```

```
vBorderColor = "Black"
```

vCellPadding = 7

```
/>
```

</form>

Note that the component resides within the form tags on the *calling* page. Also note that it sets three properties, which are vGridLines, vBorderColor, and vCellPadding.

The following list discusses the user control:

- Passing parameters from the Web Page to the Component: The User Component declares three Public variables to receive the three passed variables as follows:
 - 2. PUBLIC vGridLines As GridLines
 - 3. PUBLIC vBorderColor as String

PUBLIC vCellPadding As Integer

- 4. **Reading the XML File into a DataSet:** In the page_load event, the XML file is read and stored in a DataSet. A DataGrid is bound to the DataSet and its GridLines, BorderColor, and CellPadding attributes are set with the passed variables.
 - 5. Sub Page_Load(Source As Object, E As EventArgs)
 - 6. Dim ds As New DataSet
 - 7. Dim fs As filestream
 - 8. Dim xmLStream As StreamReader

9. fs = New filestream(Server.MapPath("nav.xml"), FileMode.Open, FileAccess.Read)

- 10. xmlStream = new StreamReader(fs)
- 11. ds.ReadXML(XmlStream)
- 12. fs.Close()

- 13. dlist.DataSource=ds.Tables("site").DefaultView
- 14. dlist.DataBind()
- 15. dlist.GridLines = vGridLines

16.

- dlist.BorderColor=System.Drawing.Color.FromName(vBorderColor)
- 17. dlist.CellPadding=vCellPadding

End Sub

18. Setting the BorderColor Property: The BorderColor property references a data type of system. To convert a string to a color I have used the FromName method. This method takes a string and converts it to type color. Thus I have declared vBorderColor as a string, such as in the following:

PUBLIC vBorderColor as string

To set the BorderColor property I have used the following code:

- dlist.BorderColor=System.Drawing.Color.FromName(vBorderColor)
- 19. <u>The DataList</u>: The DataList displays the navigation links. I have defined a hyperlink control to render the links, and this is bound to the SiteUrl tag of the XML file.

<asp:DataList runat=server id="dlist"

RepeatDirection="horizontal"

RepeatMode="Table"

Width="100%"

BorderWidth="1"

Font-Name="Verdana"

Font-Size="8pt"

HeaderStyle-BackColor="#aaaadd"

SelectedItemStyle-BackColor="yellow"

ItemStyle-BackColor="antiquewhite"

AlternatingItemStyle-BackColor="tan"

>

<ItemTemplate>

<asp:HyperLink runat="server"

Text= '<%# Container.DataItem("sitename") %>'

NavigateUrl= '<%# Container.DataItem("siteurl") %>' />

</ltemTemplate>

</asp:DataList>

You can see how simple the implementation is. Keeping our links in a separate XML file allows us to modify the Navigation structure of the Web site without having to change all the web forms.

The following is the complete listing of the user control: **nav.ascx**

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.IO" %>

<%@ Import Namespace="System.Drawing" %>

<script language="VB" runat="server">

'Public Variable for each exposed Property

PUBLIC vGridLines As GridLines

PUBLIC vBorderColor As String

PUBLIC vCellPadding As Integer

Sub Page_Load(Source As Object, E As EventArgs)

Dim ds As New DataSet

Dim fs As filestream

Dim xmLStream As StreamReader

fs = New filestream(Server.MapPath("nav.xml"), FileMode.Open, FileAccess.Read)

xmlStream = new StreamReader(fs)

ds.ReadXML(XmlStream)

fs.Close()

dlist.DataSource=ds.Tables("site").DefaultView

dlist.DataBind()

dlist.GridLines = vGridLines

dlist.BorderColor=System.Drawing.Color.FromName(vBorderColor)

dlist.CellPadding=vCellPadding

End Sub

</script>

<asp:DataList runat=server id="dlist"

RepeatDirection="horizontal"

RepeatMode="Table"

Width="100%"

BorderWidth="1"

Font-Name="Verdana"

Font-Size="8pt"

HeaderStyle-BackColor="#aaaadd"

SelectedItemStyle-BackColor="yellow"

ItemStyle-BackColor="antiquewhite"

AlternatingItemStyle-BackColor="tan"

>

<ItemTemplate>

<asp:HyperLink runat="server"

Text= '<%# Container.DataItem("sitename") %>'

NavigateUrl= '<%# Container.DataItem("siteurl") %>' />

</ltemTemplate>

</asp:DataList>

Summary

User Controls can encapsulate both presentation and code logic. In this chapter we built a very useful component that encapsulates the site navigation logic of Web site. We saw that user controls are simple to create and that any web form can be encapsulated into a user control with slight modifications.

Chapter 7: Custom Controls

Overview

Control development has often been compared to developing components for a manufactured product. Assembling a manufactured product on an assembly line involves assembling various components together to create the final product. In a similar manner, software controls are created that encapsulate generic functionality. These controls are then combined with other software components to build the final software product. The ASP.NET server controls, the DataList, the DataGrid, and the DataRepeater are examples of such software controls.

Controls have been around for a long time. We have seen a rapid proliferation of ActiveX controls and Java Applets in recent years. A major limitation in these technologies is that not all sites support these technologies. The advent of new generation products like handheld devices and cell phones make the matter more complicated. We can no longer assume that our target audience will always have the technology to support and run the controls we create. The ASP.NET server controls seek to address this problem by having the server render code as pure HTML that all browsers can understand.

ASP.NET provides a clean way of developing Custom Controls. In this chapter, I will provide you the basics of developing such controls. I will start with building a simple control, which will give an overview of the process. Then I will develop a very useful Custom Control, which is a generic add/modify control. A DataGrid does not have capabilities to add a record. In the web forms that I have built so far in this book, I have added textboxes and code to incorporate this functionality. The Custom Control that I will build will accept a SQL Query string as a property setting and will automatically generate a form to add records to the database, complete with field labels. You will pass it the name of a stored procedure and the control will call it with the required parameters to create a record. This control will also have an edit mode. You can hook the control up to a DataGrid and use it instead of the editing functionality of the DataGrid. The advantage of using this control for editing is that you do not have to code any of the events of the

DataGrid. I will introduce this topic with a simple control written first in Visual Basic and then in C#.

A Simple Control in Visual Basic

This control will simply display a "Hello World" message. Though this control is simple to create, it compiles into a working DLL and demonstrates the various steps involved in its creation. The source code for this sample can be found in the HelloWorld subfolder on the book's Web site at www.premierpressbooks.com/downloads.asp.

Step 1: Create a Class File

Using Notepad, create a Visual Basic class file and save it with the extension .vb. hello.vb

Imports System

Imports System.Web

Imports System.Web.UI

Namespace Hersh

Public Class Hello: Inherits Control

Protected Overrides Sub Render(Output As HtmlTextWriter)

Output.Write("<H1>Hello World</H1>")

End Sub

End Class

End Namespace

The following is a discussion on various aspects of the code:

 Namespaces: Namespaces is a way of grouping together related classes, interfaces, structures, enumerations, and delegates under one name. It can also be thought of as a shortcut way of referring to long names. Namespaces can be nested, and a source file can have as many namespaces as needed. In this example, the class hello resides in the namespace Hersh. In the calling web form I will register the namespace with the following directive:

<%@ Register TagPrefix="Hersh" Namespace="Hersh" Assembly="hello" %>

The control will be called as follows:

- <Hersh:Hello runat=server />
- Imports: The import directive is another shortcut that allows one to use namespaces without providing a fully qualified path. The namespace required by a server side control are System,

System.Web, System.Web.UI, and

System.Web.UI.WebControls.

- The System namespace contains core system classes and services like class activation and serialization.
- The System.Web contains services related to web services like HttpRequest and HttpResponse.
- The System.Web.UI namespace includes services like control management and cascading stylesheet management.
- The System.Web.UI.Control namespace includes the definition of the control class for building the control. The control class includes properties, methods, and events common to all server controls in an ASP.NET page. ASP.NET supports inheritance, so a new control can inherit from the control class by using the Inherits keyword. Functions in a "parent" class can be overridden to provi de specific functionality in the descendent class.
- Public: Note that I have declared the hello class to be public. A public class can be instantiated by anybody whereas a private class can only be instantiated by other classes within the same namespace.
- The Render Function: The hello class inherits from the control class, which contains the Render function. The Render function determines what our control will look like. I want to control the output of the control, so I override the function and use the write method of the HtmlTextWriter object to output the string "hello world" to the browser.

Render takes a single argument of type HtmlTextWriter, which abstracts the task of streaming HTML to the browser. This class has the functionality of automatically generating element tags, attributes, and styles. Other functions in this class are methods like WriteLineNoTabs, WriteBeginTag, and WriteEndTag to name a few.

writebegriniag, and writebinarag to r

Step 2: Create the DLL

Create a "bin" folder in wwwroot (or the root directory of your application folder) and execute the makevb.bat file (make sure that the outdir variable points to your bin folder). This file has the following commands:

makeVb.bat

set outdir=g:\AspNetSamples\bin\hello.DLL

set assemblies=System.dll,System.Web.dll

vbc /t:library /out:%outdir% /r:%assemblies% hello.vb

pause

This should create hello.dll in the bin folder. Component registration in ASP.NET has greatly improved. Previously, a component had to be registered using the registration tool (regsvr32.exe). Now all you need to do is copy the DLL into the bin folder. To update the component, just copy the revised DLL file into the bin folder. No entries are made in the registry and if you need to unregister the component, just delete it from the bin directory.

The bin is a special directory that the .NET runtime examines to identify and locate namespaces. The .NET runtime examines the metadata of assembly files placed in the bin directory from which it knows where to locate the namespace specified on the web forms. Metadata for an object records information required to use the object. Typically, this information includes the name of the object, names of all the fields of the object, and their types and details of all member functions including parameter types and names.

Assembly is the method of packaging all the files required for an object in one complete package. The .NET compiler compiles code to an intermediate form called "IL" (intermediate language). The assembly contains "IL," metadata, and other files in one comprehensive package. Each application can have its own bin folder. The components present here are local to that application.

Step 3: Create the Web Form hellovb.aspx

<%@ Register TagPrefix="Hersh" Namespace="Hersh" Assembly="hello" %>

<html>

<body>

```
<Hersh:Hello runat=server />
```

</body>

</html>

I have used the Register directive to register the functionality existing in the namespace Hersh. Now, it is a simple task to initiate the class by using the tag <Hersh:Hello runat=server />.

Creating a Simple Control in C#

I will now discuss building a control in C#. This control will also render "Hello World" to the browser. However, this time I will define a property called message, which I will use to display the string "Hello World". The control will also display the system date and time. The source code for this sample can be found in the HelloWorld subfolder on the book's Web site at www.premierpressbooks.com/downloads.asp.

Step 1: Create a Class File helloC.cs

using System;

```
using System.Web.UI;
using System.ComponentModel;
namespace CustomControls
{
 public class FirstC : Control
 {
  private String message = "";
  public virtual String Message
  {
   get
    {
     return message;
   }
   set
    {
     message = value;
    }
  }
      protected override void Render( HtmlTextWriter writer)
  {
  writer.Write("<font> "+ this.Message + "<br>" +
      "The server date and time : " +
       System.DateTime.Now +
    "</font>");
  }
 }
```

This code is almost similar to the Visual Basic code. There are slight syntax differences, which are noted in the following:

1. "Using" instead of "Imports" Visual Basic: Imports System C#: using System; 2. NameSpace Visual Basic: NameSpace Hersh 'Some Code here End NameSpace C#: NameSpace{ //Some code here } 3. Inherits Visual Basic: Public Class Hello: Inherits Control C#: public class FirstC : Control

A property called message is defined as that which has get and set assessor methods. The string "Hello World" is passed using this property.

Step 2: Create the DLL

The following commands compile the control to a dll: **makec.bat**

set outdir=G:\AspNetSamples\bin\helloc.DLL

set assemblies=System.Web.dll

csc /t:library /out:%outdir% /r:%assemblies% helloC.cs

pause

A control called helloc.dll should be created in the bin directory.

Step 3: Create the Web Form

Now create the web form that will call this control as follows: HelloC.aspx

<%@ Register TagPrefix="Custom" Namespace="CustomControls" Assembly="helloc"%>

<html>

<body>

<form method="POST" runat=server>

<Custom:FirstC Message= "Hello World in C#" runat=server/>

</form>

</body>

</html>

The control resides in the namespace CustomControls. The following initiates the control and passes it the string "Hello World" in C#":

<Custom:FirstC Message= "Hello World in C#" runat=server/>

The Generic Edit/Add Custom Control

There is a history behind this control. In ASP, there was no easy way to display database data in the browser. We had to write code to open a database connection, feed a recordset with the data from the database, and then iterate the recordset to render the data as HTML. Many brains in the ASP community set about to develop generic routines to automate the task of rendering database data to the browser. Alan Saldanha wrote a primer in which he described the design of such a generic tool. Eli Robillard built a brilliant tool based on his theory and made it available as freeware on his site (http://www.ofifc.org/Eli/ASP/GenericArticle.asp). Eli named this tool Genericdb and it was comprised of a series of ASP pages. All you had to do to output a database table (or the result of a SQL query) to the browser was to build a "config" file in which you specified various variables like the table name, the DSN, the display fields, etc., and a beautiful grid-like table was created. This table looked like the DataGrid in ASP.NET and like it, had alternating color style, paging, sorting, etc. Eli called this page the "Lister." This grid had three links: add, delete, and modify. Clicking on these links allowed you to add, update, or delete a record. Roman Koch developed EDITOR.ASP (http://www.4guysfromrolla.com/webtech/110999-1.shtml) after he saw Genericdb.

This was a "no-frills" version on Genericdb, written as a series of function calls. He was able to pack a basic "Lister" and "Editor" into a single 10k file.

The ASP.NET DataGrid provides the "Lister" functionality of the Genericdb. In other words, it displays a list of records that can then be modified by clicking on an edit link. The Edit mode of the DataGrid is equivalent to the "Editor" functionality of Genericdb. However, this functionality is not automatic. You need to code a number of DataGrid events (for example, the OnEditCommand, OnUpdateCommand, OnCancelCommand, etc.), read the modified values, and send the updated values back to the database. Another drawback of the Edit mode is that the layout of the grid in this mode is horizontal. We might want a vertical layout, which is more common in web forms.

There is no "insert" mode in the DataGrid. If you want to incorporate "insert" functionality in your Web pages you must fall back to the traditional method of building a user input form by adding textboxes on the page. These considerations led me to write the GenEditAdd custom control. This control can be hooked up to a DataGrid, and it provides both the "edit" and "insert" functionality in a consistent manner. You do not have to code any events; just set a few properties and you are done. I suggest you compile and experiment with the GenEditAdd control first. You will find the sample files for this control in the GenEditAdd_final subfolder on the book's Web site at www.premierpressbooks.com/downloads.asp. I will go into a detailed discussion of it later. You will need the Masters table in your database to test it out. If you have not set

up the example database, do so now as specified in <u>Appendix A</u>. The following is how you test it:

- Run the bat file mGenEditAdd.bat: This will compile and put the control (GenEditAdd.DLL) in the bin folder under wwwroot. Be sure to modify this file to point to your bin folder
- 2. Open masters.aspx and run it through IIS.

The masters.aspx form contains a DataGrid. Just look at the visual appearance at the moment. I will discuss the code later. This web form displays all the records from the Masters table. There are three links attached to each row. These are the add, edit, and delete links. Figure 7.1 shows what it looks like.

Ex ' i	0	Petroh	Fione	Search	favoite:	Hotoy	Mai Na	- Pex	E.	Meconor
hart of Acces	unts:									
Delete	Account	5.0	Group	-11 C	Тури	0 Open	ng Cleain			
Edit Add Delete	Cach in Hand		Cash in he	nd	λ	0	0			
Lot Add Celets	Wells Farge Chi	eching	Cesh in he	nd	A		9			
Colt Add Celeta	Vise Card		Cesh is he	nd	A		700			
Edit Add Delete	Salary		Sales acco	UNI:	1		0			
Edit Add Celete	Interest Income	HE	Sales acco	iant	1		400			
Edit Add Celeta	Rant		Furchase a	scouurt	1		3			
Edit Add Celeta	USINER		Furchess a	scourt	5		0			
Edit Add Celets	Medical Expensi	P5	Furchase i	transit	E		500			
ticit Add Celets	hersh		Functions account		t.		0			
Loit Add Celete	hansh		Purchase a	scourt	1		0			
Lot Add Celeta	hanahohasin 1		Purchase a	popeert	E		0			
	DOP		Dark exce	deu	A		400			

Figure 7.1: The Lister.

Clicking on the edit link activates the GenEditAdd control in the Edit mode. I can make modifications and hit the update button. This will save the changes to the database. Figure 7.2 shows the GenEditAdd control in Edit mode.



#State Whose States Elect. Elect. Black YNMESSING #STES 205H

Figure 7.2: The GenEditAdd control in the Edit mode.

Clicking the add link activates the control in the Add mode. Figure 7.3 shows the GenEditAdd control in Add mode.

giden @ ht	p.//127.00.1/MS	PEquilW	p/Chapte?/	GerEd&Add	Voorlig_med	ier.aspe?cod	C-suler_si				-	200
Ģ Back	- \$ 1999	0 500	Petroh	Hone	G. Snarch	Favoites	C History	NN.	-Ber	The second	Meconger	8
lack												
Add a New	Record:											
code_display	Test											
ode_catego	1											
spe												
tspe closing Add Case gcode_vab	Taecute p e=NULL	_marte	rı Brode	_display=	Te#, @:	oode_categ	po g=1, @	type=", @	closing=1	10,		
sse kosing Add Cane Brode_vab	Tancate p e=NULL	_marte	rı Əsok	_daplay=	Test, இ	code_cates	on=1.@	tyye ~* .@	(closing=)	10,		

Figure 7.3: The GenEditAdd control in the Add mode.

The control automatically builds a procedure call to the stored procedure p_masters and passes it the appropriate parameters. I will discuss this procedure in <u>Chapter 15</u>, "<u>Chart of Accounts.</u>" For now, know that this procedure is responsible for both inserting and updating a masters record. If we need to modify a record, I send it the primary key of the record to be modified as a parameter. The code_value is the primary key of the Masters table. Thus, to update a record with code_value of 3, I call the stored procedure as follows:

Execute p_masters @code_display='Visa Card ', @code_category=604, @type='A', @closing=700, @code_value=3

If we want to insert a new record, I send a null value as the code_value to this procedure as follows:

Execute p_masters @code_display='Test', @code_category=1, @type='', @closing=10, @code_value=NULL

I do not have to manually build this string as the GenEditAdd control reads the user input to extract the input values, as well as the database fields collection to get the column names. It automatically builds this procedure call string, and posts it to the database.

The Config File

The DataGrid in the Masters web form has two hyperlink columns: one for the edit link and the other for the add link. These links navigate to the config_masters.aspx form. The edit link passes it the primary key of the record (code_value in this case) as follows:

<asp:HyperLinkColumn Text="Edit" DataNavigateUrlField="code_value" DataNavigateUrlFormatString="config_masters.aspx?code_value={0}"/> The add link passes it a code_value of zero as follows:

<asp:HyperLinkColumn Text="Add" DataNavigateUrlField="code_value" DataNavigateUrlFormatString="config_masters.aspx?code_value=0"/> The config_masters.aspx is the form where the GenEditAdd control resides. Each DataGrid that wants the edit and add functionality will define a separate config form. This form contains the GenEditAdd control and a number of property settings for this component. The code_value passed to this form is extracted in the page load event. Various properties are also set here. Take a look at the following list:

The Config_Masters.aspx form

<%@ Register TagPrefix="Hersh" Namespace="Generic_chap7" Assembly="GenEditAdd_Chap7" %>

<html>

```
<script language="VB" runat="server">
```

Sub page_load(sender As Object, e As EventArgs)

if NOT (isPostBack)

Dim sql As string

Dim Is_CodeValue As string

ls_CodeValue = Request.QueryString("code_value")

SQL = "Select * from masters"

Gen.sql = SQL

if cint(ls_codeValue) = 0 then

Gen.Where = ""

else

Gen.where= " Where code_value =" + Is_CodeValue

end if

Gen.display = "111110"

Gen.KeyField = "code_value"

Gen.KeyValue = ls_codeValue

Gen.procedure = "p_masters"

```
Gen.ExitPage = "masters.aspx"
```

end if

End Sub

```
</script>
```

<body>

<form runat = "server" >

<Hersh:GenEditAdd_Chap7 id = "Gen" runat=server

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;" />

</form>

</body>

</html>

In this form, the GenEditAdd control is first registered with the following page directive:

<%@ Register TagPrefix="Hersh" Namespace="Generic_chap7" Assembly="GenEditAdd_Chap7" %>

The control is then created with the id of Gen.

<Hersh:GenEditAdd_Chap7 id = "Gen" runat=server ConnStr = "Provider=SQLOLEDB;</pre> Data Source=(local); Initial Catalog=ASPNET;User ID=sa;" />

The GenEditAdd control requires certain properties to be set and these are set in the Page_Load event. The following tables show the properties of the GenEdit-Add component:

1. Property	SQL
Meaning	The SQL String without the Where Clause
Example	Select * from Masters
2. Property	Where
Meaning	If the Where property is provided, GenEditAdd displays the Edit mode. If it is blank, it displays the insert mode. The where clause is built dynamically in the config form based on the passed code_value as follows: Is_CodeValue = Request.QueryString("code_v alue") SQL = "Select * from masters" Gen.sql = SQL If cint(Is_codeValue) = 0 Then Gen.Where = "" Else Gen.where= "Where code_value =" + Is_CodeValue End If If a code_value of zero is passed in the query string, the Where property is set to blank, thus displaying the control in the insert mode. If the Where property is provided, a where clause is built and the control displays in the Edit mode.
3. Property	Display
Meaning	This is a string of 0s and 1s. Zero means don't show a field and 1 means show it. Say the masters table has four fields. The string 0101 would mean hide the first and third fields and

1. Property	SQL
	show the second and forth.
Example	0101
4. Property	KeyField
Meaning	The primary key field name
Example	code_value
5. Property	KeyValue
Meaning	The value of the primary key
Example	2 (this will be passed to the config form from the calling form)
6. Property	Procedure
Meaning	The stored procedure to be called for inserting/updating a record
Example	p_masters
7. Property	ExitPage
Meaning	This creates a hyperlink on the top of the edit/add form which allows you to navigate back to the calling form.
Example	masters.aspx
8. Property	ConnStr
Meaning	The Connection String
Example	"Provider=SQLOLEDB; Data Source=(local); Initial _ Catalog=ASPNET;User ID=sa;"

As you can see there are eight properties that need to be set for the GenEditAdd component. The code of the control resides in the file GenEditAdd.vb. I will now start building it and discuss the theory of building custom controls as I go along.

Building the Control

The complete source code for this custom control resides in the file GenEdit-Add_chap7.vb. There is a fair amount of code in this file and some theory that needs to be explained. I have thus decided to break the topic in a series of steps. I will be building intermediate code files, compiling it, explaining some aspects of the control until I build the final control in the file GenEditAdd_chap7.vb. You will find the code for these steps in the subfolder called "steps" on book's Web site at www.premierpressbooks.com/downloads.asp.

www.preimerpressbooks.com/download

Step 1: The Edit Mode

I have not started building the control in this step. Step1.aspx is a simple aspx form that shows how I build a user input form in the Edit mode of the control. For GenEditAdd to

display in Edit mode, we must pass it a SQL string as well as a "Where" clause. This is hardcoded in this web form, but will be converted to properties in the GenEditAdd control. Figure 7.4 shows the output generated by the form.



<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

Step1.aspx

<script language="VB" runat="server">

Sub Page_Load(sender As Object, e As EventArgs)

Dim dv As DataView

Dim i As integer

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Dim Where As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

SQL = "select * from groups "

Where = "where code_value = 700"

myCommand = New OleDbDataAdapter(SQL + Where, myConnection) myCommand.Fill(ds, "groups") dv = new DataView(ds.Tables("groups")) Dim t As DataTable t = dv.Table Dim r As DataRow Dim c As DataColumn Dim cell As TableCell Dim row As DataRow For Each r in t.Rows For Each c in t.Columns response.write(c.ToString + ": ")

response.write(r(c).ToString + "
")

Next c

Next r

End Sub

</script>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<asp:Table id="Table" Font-Name="Verdana" Font-Size="8pt" CellPadding=5 CellSpacing=0 BorderColor="black"

BorderWidth="1" Gridlines="Both" runat="server"/>

</body>

</html>

Note that I have a SQL statement and a "Where" clause as follows:

SQL = "select * from groups "

Where = "where code_value = 700"

I load a DataSet with the data returned from this query and assign it to a DataView. This DataView is assigned to a DataTable, which allows me to iterate the DataColumn collection (the inner loop) for each DataRow (the outer loop) as follows:

dv = new DataView(ds.Tables("groups"))

Dim t As DataTable

```
t = dv.Table
Dim r As DataRow
Dim c As DataColumn
Dim cell as TableCell
Dim row as DataRow
For Each r in t.Rows
For Each c in t.Columns
   response.write(c.ToString() + ": ")
   response.write(r(c).ToString() + "<br>>")
Next c
```

Next r

I get the column name (c.ToString) and will display it as a label in the GenEdit-Add control. I obtain the value of the column (by r(c).ToString) and in the GenEditAdd control, which we will build in Step 3, this will be displayed inside a textbox which can be modified by the user.

Step 2: The Add Mode

I have still not started building the control. Step2.aspx is a web form that explains the code in the add mode. In this mode, I do not supply the "Where" clause. The "SQL" clause is still required, as I must read the columns collection to extract the column names and display them as the labels against fields. Here is Step2.aspx.

Step2.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server">

Sub Page_Load(sender As Object, e As EventArgs)

Dim dv As DataView

Dim i As integer

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

SQL = "select * from groups " myCommand = New OleDbDataAdapter(SQL, myConnection) myCommand.Fill(ds, "groups") dv = new DataView(ds.Tables("groups")) Dim t As DataTable t = dv.Table Dim r As DataRow Dim c As DataColumn Dim cell as TableCell Dim row as DataRow

For Each c in t.Columns

response.write(c.ToString() + "
 ")

Next c

End Sub

</script>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<asp:Table id="Table" Font-Name="Verdana" Font-Size="8pt" CellPadding=5 CellSpacing=0 BorderColor="black"

BorderWidth="1" Gridlines="Both" runat="server"/>

</body>

</html>

In the insert mode, I need only iterate the DataColumns collection. Figure 7.5 shows the output from this form.



Figure 7.5: The Add Mode.

Step 3: The First Build

We are now ready to start building the control. Before we do that, let's build a web form, which will allow us to test the control as we are building it. The code for this step can be found in the ...GenEditAdd\Steps\step3 subfolder on the book's Web site at <u>www.premierpressbooks.com/downloads.asp</u>.

GenTestStep3.aspx

<%@ Register TagPrefix="Hersh" Namespace="Generic_chap7_step3" Assembly="GenEditAdd_Chap7_step3" %>

<html>

```
<script language="VB" runat="server">
```

Sub page_load(sender As Object, e As EventArgs)

if NOT (isPostBack)

Dim vsql As string

Gen.sql = "select * from masters"

Gen.where= " Where code_value = 1"

Gen.display = "111110"

Gen.KeyField = "code_value"

Gen.KeyValue = "1"

Gen.procedure = "p_masters"

Gen.ExitPage = "GenTestStep3.aspx"

end if

End Sub

</script>

<body>

<form runat = "server" >

<Hersh:GenEditAdd_Chap7 id = "Gen" runat=server</pre>

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;" />

</form>

</body>

</html>

S

This web form simply initiates the GenEditAdd control and sets its eight properties. The user control will display in the Edit mode as we are passing it a "where" clause. To display it in the add mode, just pass a blank value for the where clause. In this step, we make the first build of the control. The code resides in the file Step3.vb and it should be compiled by running Step3.bat. Edit the bat file so that the outdir variable points to your bin folder. I shall list out the code file and then discuss it.

tep3.vb			

Option Strict Off

Imports System

Imports System.Web

Imports System.Web.UI

Imports System.Web.UI.WebControls

Imports System.Data

Imports System.Data.OleDb

Namespace Generic_Chap7_step3

Public Class GenEditAdd_Chap7 : Inherits Control : Implements INamingContainer

Private Is_display as string

Private Is_where as string

Private Is_sql as string

Private Is_ConnStr as string

Private Is_keyField as string

Private Is_keyValue as string

Private Is_procedure as string Private Is_exitpage as string Private It_datatable as datatable Private Is_mode as string Protected mytbl as table Public Property Mode as string Get Return Cstr(ViewState("ls_mode")) End Get Set ViewState("Is_mode") = value End Set End Property Public Property ExitPage as string Get Return Cstr(ViewState("ls_exitpage")) End Get Set ViewState("Is_exitpage") = value End Set End Property Public Property t as datatable Get Return It_datatable End Get Set lt_datatable = value End Set End Property Public Property KeyField as string

Get Return Cstr(ViewState("ls_keyfield")) End Get Set ViewState("ls_keyfield") = value End Set End Property Public Property KeyValue as string Get Return Cstr(ViewState("ls_keyvalue")) End Get Set ViewState("ls_keyvalue") = value End Set End Property Public Property Procedure as string Get Return Cstr(ViewState("ls_procedure")) End Get Set ViewState("Is_procedure") = value End Set End Property Public Property display as string Get Return Cstr(ViewState("ls_display")) End Get Set ViewState("Is_display") = value End Set

End Property Public Property Where as string Get Return Cstr(ViewState("ls_where")) End Get Set ViewState("ls_where") = value End Set End Property Public Property SQL as string Get Return Cstr(ViewState("ls_sql")) End Get Set ViewState("ls_sql") = value End Set End Property Public Property ConnStr as string Get Return Cstr(ViewState("Is_ConnStr")) End Get Set ViewState("ls_ConnStr") = value End Set End Property Protected Overrides Sub CreateChildControls() Dim dv As DataView Dim myConnection As OleDbConnection Dim myCommand As OleDbDataAdapter Dim ds As New DataSet

```
Dim vSql As string
   If Where.Length < 1 then
    vSql = SQL
    mode = "insert"
   Else
    vSql = SQL + Where
    mode = "update"
   End If
   myConnection = New OleDbConnection(ConnStr)
   myCommand = New OleDbDataAdapter(vSql, myConnection)
   myCommand.Fill(ds, "vtable")
   dv = new DataView(ds.Tables("vtable"))
   Dim Fields As Integer
   'Dim t As DataTable
   t = dv.Table
   Dim r As DataRow
    Dim c As DataColumn
   Dim cell As TableCell
   Dim row As DataRow
   Dim Fieldscount As integer
   Dim s As string
   Dim vdisplay As string
   FieldsCount = 0
   s = "<A HREF=" + ExitPage + ">Back</A>"
   me.Controls.Add(new LiteralControl(s))
   me.Controls.Add(new LiteralControl("<table bgcolor ='antiquewhite' style='font: 8pt
verdana'>"))
   me.Controls.Add(new LiteralControl(""))
```

If mode = "insert" then

me.Controls.Add(new LiteralControl("Add a New

Record:"))

Else

me.Controls.Add(new LiteralControl("Edit

Record:"))

End if

me.Controls.Add(new LiteralControl(""))

If mode = "update" then

For Each r in t.Rows

For Each c in t.Columns

'Don't show this field

IF vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField then

Else

me.Controls.Add(new LiteralControl(""))

'label

me.Controls.Add(new LiteralControl(""))

me.Controls.Add(new LiteralControl(c.ToString))

me.Controls.Add(new LiteralControl(""))

'value

me.Controls.Add(new LiteralControl(""))

Dim Box As New TextBox

Box.Text = r(c).ToString

Box.ID = c.ToString

me.Controls.Add(box)

me.Controls.Add(new LiteralControl(""))

end if

FieldsCount = FieldsCount + 1

Next c

Next r

Else ' Insert Mode

For Each c in t.Columns IF vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField then Else me.Controls.Add(new LiteralControl("")) 'label me.Controls.Add(new LiteralControl("")) me.Controls.Add(new LiteralControl(c.ToString)) me.Controls.Add(new LiteralControl("")) 'value me.Controls.Add(new LiteralControl("")) Dim Box As New TextBox Box.ID = c.ToString me.Controls.Add(box) me.Controls.Add(new LiteralControl("")) End if FieldsCount = FieldsCount + 1 Next c End If me.Controls.Add(new LiteralControl("")) me.Controls.Add(new LiteralControl("</Table>")) '----add button Dim AddButton As New Button if mode = "insert" then AddButton.Text = "Add" else

AddButton.Text = "Update"

end if

Me.Controls.Add(AddButton)

End Sub

End Class

End Namespace

Properties and Namespaces

I have imported various namespaces in my component. These are the following:

- System
- System.Web
- System.Web.UI
- System.Web.UI.WebControls
- System.Data
- System.Data.ADO

Defining properties for the control is a neat way of allowing users to supply desired values to the control. Properties are implemented by defining accessor (get) and mutator (set) functions and a local variable. The GenEditAdd control uses eight property values as discussed above and two properties that are of internal use. Thus there are a total of ten property values.

I have thus declared ten local variables as follows:

Private Is_display as string

Private Is_where as string

Private Is_sql as string

Private Is_ConnStr as string

Private Is_keyField as string

Private Is_keyValue as string

Private ls_procedure as string

Private Is_exitpage as string

Private It_datatable as datatable

Private Is_mode as string

The ten properties are as follows:

- Display
- Where
- SQL
- ConnStr
- KeyField
- KeyValue
- Procedure
 ExitPage
- ExitPageDataTable
- Data rable
 Mode (insert/edit)

The DataTable and Mode properties are used internally by the control and the user does not set them. You will note that I have used the ViewState property with each of these. Thus the SQL property looks like the following:

Public Property SQL as string

Get

Return Cstr(ViewState("ls_sql"))

End Get

Set

ViewState("ls_sql") = value

End Set

End Property

The properties that are set by the users are stored in a "StateBag." The ASP.NET framework will save the state of the control when it is destroyed and restore it when it is created. The <code>ViewState</code> property gets the stored property from the state bag. If we don't use the state property, we will lose the property values each time a post-back occurs.

Control Composition

The simple control that I created in the start of this chapter is called a "Noncomposite Control." This was because I had to control the rendering of HTML to the browser, which I did by overriding the control's Render method and taking charge of the rendering. The GenEditAdd is called a composite control because it is composed of standard ASP.NET controls like textboxes and buttons.

The GenEditAdd control inherits from the control class. This class has an overrideable method called CreateChildControls. The purpose of this method is to create child controls on the form. There are two types of controls that I add to the form. These are LiteralControls and TextBoxes. Anytime I want to add a label or write out a string containing HTML formatting, I use the LiteralControl. For example, to add a starting table tag, I add the following LiteralControl using the method called controls.Add:

me.Controls.Add(new LiteralControl(""))

I also add a number of textboxes to build the input form. I will be coming back to this later. I identify the mode the control is in. This can be the update or insert mode and the property called Mode is updated accordingly. This is done by the following code:

If Where.Length < 1 then

vSql = SQL mode = "insert" Else vSql = SQL + Where

mode = "update"

End If

If the Where property is less than one character, the Mode property is set to "insert," else the Mode property is "update" and the where clause is added to the SQL statement. If the Mode property is update, I iterate the DataRows and DataColumns collections as explained in Step 1, otherwise I just iterate the DataColumns collection as explained in Step 2. A number of textboxes are added to the form to receive the user input as follows:

Dim Box As New TextBox

Box.Text = r(c).ToString

Box.ID = c.ToString

me.Controls.Add(box)

Note that I give each textbox a unique id so that I can refer to it in code. This id is the name of the database column. The textboxes are enclosed within table data () tags. The table (), table row (), and table data () tags are all created using LiteralControls.

I can specify the fields to display by setting the Display property. Suppose my database fields are code_value, code_display, type, opening, and closing. I do not want to display the code_value field as I do not want the user to be able to change the primary key. Also, I do not want to display the type field (this is set automatically by the stored procedure p_masters and the user should not be allowed to change it) and the

closing balance field (a database trigger will automatically update this field. This trigger, which is on the transactions table, will be explained later in the book). Hence, I will set the Display property to be 01010. Remember, a zero means hide the field and a one means show it. In the body of the control, I pad the Display property with a number of zeros as follows:

If vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField Then

'Do Nothing

Else

'Proceed

End if

The chars method extracts the value of a string at a specified position. This position is provided by the FieldsCount variable, which I am incrementing in the loop. If the extracted character at the position provided by the FieldsCount variable is equal to zero or the column is the KeyColumn, then the control doesn't do anything. Otherwise, it creates a textbox with the appropriate id. Finally, I add a button that will display with a caption of "Update" in the edit mode and "Insert" in the insert mode.

The InamingContainer

Run the test web form GenTestStep3.aspx. View the source in the browser. The following is an extract of what you should see:

```
<A href=>Back</A>
```

<Table bgcolor ='antiquewhite' style='font: 8pt verdana'>

```
Edit Record:
code display
 <input name="Gen:code_display" type="text" value="Cash in Hand
"id="Gen_code_display" />
  code_category
 <input name="Gen:code_category" type="text" value="604" id="Gen_code_category"
/>
 type
```

<input type="submit" name="Gen:ctrl31" value="Update" />

Note that each textbox has an id attribute. For example, the code_display has an id of Gen_code_display. I had supplied the id of code_display in the body of the code. However, the name Gen was provided by the control on its own accord. This means all the textboxes (each having a unique id) reside within the parent control called Gen, and thus have this name prefixed to its id.

I told the ASP.NET runtime that GenEditAdd was an InamingContainer when I said that it Implements InamingContainer at the top of my code file. This is all that I need to do for the control to participate in ID and state management.

To see this for yourself, delete the Implements InamingContainer directive so that the code now says the following:

Public Class GenEditAdd: Inherits Control

Compile Step3.vb (by running step3.bat). Now run GenTestStep3.aspx and view the source of the generated HTML, as follows:

Back

Edit Record:

```
code_display
```

<input name="code_display" type="text" value="Cash in Hand " id="code_display"

/>

 code_category

```
<input name="code_category" type="text" value="604" id="code_category" />
  type
  <input name="type" type="text" value="A" id="type" />
  closing
  <input name="closing" type="text" value="0" id="closing" />
  </Table>
```

<input type="submit" name="ctrl37" value="Update" /> Note that now you don't see the prefix of Gen before the textbox ids. If you make any changes to the input fields and hit "Update" you lose your changes when a post-back occurs (i.e., the state is not maintained now).

Step 4: GenEditAdd Custom Control

In this step, I wrap up the GenEditAdd control. The final code contained is in the file GenEditAdd.vb. You will find the source files for this step in the ...GenEditAdd\Steps\Step4 subfolder on the book's Web site at <u>www.premierpressbooks.com/downloads.asp</u>. Compile the GenEditAdd component by running the bat file mGenEditAdd.bat.

Handling Events

The GenEditAdd control contains a button called AddButton. When this button is clicked, I need to make a call to my stored procedure, sending it the changed/new values so that it can update or insert a new record. I code an event Handler to do this job.

EventHandlers are attached to a button by attaching delegates to the events raised by the child controls. I have done this for the AddButton control as follows:

AddHandler AddButton.Click, AddressOf AddBtn_Click

All this is telling the ASP.NET runtime is that when the AddButton is clicked, fire off the method AddBtn_click. This method contains the code required to build the procedure call. The following is the method:

Private Sub AddBtn_Click(Sender As Object, E As EventArgs)

'Build the procedure call

Dim s As String

Dim r As DataRow

Dim c As DataColumn

Dim cell As TableCell

```
Dim row As DataRow
Dim column As string
Dim Value As string
Dim Fieldscount As integer
Dim vdisplay As string
FieldsCount = 0
s = "Execute " + procedure + ""
If mode = "update" Then
 For Each r in t.Rows
  For Each c in t.Columns
   If vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField Then
   Else
    Dim tb As TextBox
    tb = me.FindControl(c.ToString)
    column = c.ToString
    Value = tb.text
    If c.DataType.ToString = "System.String" Then
     s = s + " @" + column + "='" + value + "', "
    Else
     s = s + " @" + column + "=" + value + ", "
    End If
   End If
   FieldsCount = FieldsCount + 1
  Next c
 Next r
 s = s + "@" + KeyField + "=" + KeyValue
 me.Controls.Add(new LiteralControl(s))
 RunSql(s)
Else
 For Each c in t.Columns
  If vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField Then
  Else
   Dim tb As TextBox
   tb = me.FindControl(c.ToString)
   column = c.ToString
    Value = tb.text
   If c.DataType.ToString = "System.String" Then
    s = s + " @" + column + "='" + value + "', "
   Else
     s = s + " @" + column + "=" + value + ", "
   End If
  End If
```

```
FieldsCount = FieldsCount + 1
Next c
s = s + "@" + KeyField + "=NULL"
me.Controls.Add(new LiteralControl(s))
RunSql(s)
```

End If

End Sub

The code in this event handler follows similar logic regarding the insert and update modes as discussed in Steps 1 and 2. A procedure call string is built and depending on the mode, the code_value parameter is a null (insert mode) or equals the primary key of the record that needs to be updated (update mode). The string that is built is written out on the screen as a LiteralControl, so that you can see what is being submitted to the database. Finally, the string is passed on to our old friend RunSql, which does the actual job of interacting with the database. The following is the RunSql method:

Sub RunSql(vSql as string)

try

Dim s As string

Dim myConnection As OleDbConnection

myConnection = New OleDbConnection(ConnStr)

Dim mycommand As New OleDbCommand(vsql,myConnection)

myconnection.Open()

myCommand.ExecuteNonQuery()

myconnection.Close()

Catch ex As OleDbException

' SQL error

Dim errItem As OleDbError

Dim errString As String

Dim s As string

For Each errItem In ex.Errors

errString += ex.Message + "
>"

Next

s = "
sqL Error.Details follow:
" & errString

me.Controls.Add(new LiteralControl(s))

Catch myException as Exception

me.Controls.Add(new LiteralControl("Exception: " + myException.ToString()))

End try

End Sub

I have included two aspx forms that you can use to test out the control in the insert and update modes. These forms are called GenTestStep4_insert.aspx and GenTestStep4_update.aspx.

This concludes the discussion on the GenEditAdd Custom Control. Here is the complete code listing for your viewing pleasure: GenEditAdd.vb **Option Strict Off** Imports System Imports System.Web Imports System.Web.UI Imports System.Web.UI.WebControls Imports System.Data Imports System.Data.OleDb Namespace Generic_Chap7 Public Class GenEditAdd_Chap7 : Inherits Control : Implements INamingContainer Private Is_display as string Private Is_where as string Private Is_sql as string Private Is_ConnStr as string Private Is_keyField as string Private Is_keyValue as string Private Is_procedure as string Private Is_exitpage as string Private It_datatable as datatable Private Is_mode as string Protected mytbl as table Public Property Mode as string Get Return Cstr(ViewState("ls_mode")) End Get Set ViewState("ls_mode") = value End Set End Property Public Property ExitPage as string Get
Return Cstr(ViewState("ls_exitpage")) End Get Set ViewState("Is_exitpage") = value End Set End Property Public Property t as datatable Get Return It_datatable End Get Set It datatable = value End Set End Property Public Property KeyField as string Get Return Cstr(ViewState("ls_keyfield")) End Get Set ViewState("ls_keyfield") = value End Set End Property Public Property KeyValue as string Get Return Cstr(ViewState("ls_keyvalue")) End Get Set ViewState("ls_keyvalue") = value End Set End Property

Public Property Procedure as string Get Return Cstr(ViewState("ls_procedure")) End Get Set ViewState("Is_procedure") = value End Set End Property Public Property display as string Get Return Cstr(ViewState("ls_display")) End Get Set ViewState("Is_display") = value End Set End Property Public Property Where as string Get Return Cstr(ViewState("ls_where")) End Get Set ViewState("ls_where") = value End Set End Property Public Property SQL as string Get Return Cstr(ViewState("ls_sql")) End Get Set ViewState("ls_sql") = value

End Set

End Property

Public Property ConnStr as string

Get

Return Cstr(ViewState("ls_ConnStr"))

End Get

Set

ViewState("ls_ConnStr") = value

End Set

End Property

Protected Overrides Sub CreateChildControls()

Dim dv As DataView

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim vSql As string

If Where.Length < 1 then

vSql = SQL

mode = "insert"

Else

```
vSql = SQL + Where
```

mode = "update"

End If

myConnection = New OleDbConnection(ConnStr)

myCommand = New OleDbDataAdapter(vSql, myConnection)

myCommand.Fill(ds, "vtable")

dv = new DataView(ds.Tables("vtable"))

Dim Fields As Integer

'Dim t As DataTable

t = dv.Table

Dim r As DataRow

Dim c As DataColumn

Dim cell As TableCell

Dim row As DataRow

Dim Fieldscount As integer

Dim s As string

Dim vdisplay as string

FieldsCount = 0

s = "Back"

me.Controls.Add(new LiteralControl(s))

me.Controls.Add(new LiteralControl("verdana'>"))

```
me.Controls.Add(new LiteralControl(""))
```

If mode = "insert" then

me.Controls.Add(new LiteralControl("Add a New

Record:"))

Else

me.Controls.Add(new LiteralControl("Edit

Record:"))

End if

me.Controls.Add(new LiteralControl(""))

If mode = "update" then

For Each r in t.Rows

For Each c in t.Columns

'Don't show this field

IF vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField then

Else

me.Controls.Add(new LiteralControl(""))

'label

me.Controls.Add(new LiteralControl("")) me.Controls.Add(new LiteralControl(c.ToString)) me.Controls.Add(new LiteralControl("")) 'value me.Controls.Add(new LiteralControl("")) Dim Box As New TextBox Box.Text = r(c).ToStringBox.ID = c.ToString me.Controls.Add(box) me.Controls.Add(new LiteralControl("")) end if FieldsCount = FieldsCount + 1 Next c Next r Else ' Insert Mode For Each c in t.Columns IF vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField then Else me.Controls.Add(new LiteralControl("")) 'label me.Controls.Add(new LiteralControl("")) me.Controls.Add(new LiteralControl(c.ToString)) me.Controls.Add(new LiteralControl("")) 'value me.Controls.Add(new LiteralControl("")) Dim Box As New TextBox Box.ID = c.ToString me.Controls.Add(box) me.Controls.Add(new LiteralControl(""))

```
End if
   FieldsCount = FieldsCount + 1
  Next c
 End If
 me.Controls.Add(new LiteralControl(""))
 me.Controls.Add(new LiteralControl("</Table>"))
 '----add button
 Dim AddButton As New Button
 if mode = "insert" then
  AddButton.Text = "Add"
 else
  AddButton.Text = "Update"
 end if
 AddHandler AddButton.Click, AddressOf AddBtn_Click
 Me.Controls.Add(AddButton)
 Dim cancel As New Button
 cancel.Text = "Cancel"
 Me.Controls.Add(cancel)
End Sub
Private Sub AddBtn_Click(Sender As Object, E As EventArgs)
 'Build the procedure call
 Dim s As String
 Dim r As DataRow
 Dim c As DataColumn
 Dim cell As TableCell
 Dim row As DataRow
 Dim column As string
 Dim Value As string
 Dim Fieldscount As integer
 Dim vdisplay As string
```

FieldsCount = 0

s = "Execute " + procedure + ""

If mode = "update" then

For Each r in t.Rows

For Each c in t.Columns

```
IF vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField then
```

Else

Dim tb As TextBox

tb = me.FindControl(c.ToString)

column = c.ToString

Value = tb.text

If c.DataType.ToString = "System.String" Then

```
s = s + " @" + column + "='" + value + "', "
```

Else

```
s = s + " @ " + column + "=" + value + ", "
```

End If

End IF

```
FieldsCount = FieldsCount + 1
```

Next c

Next r

```
s = s + "@" + KeyField + "=" + KeyValue
```

me.Controls.Add(new LiteralControl(s))

RunSql(s)

Else

For Each c in t.Columns

IF vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField then

Else

Dim tb As TextBox

tb = me.FindControl(c.ToString)

```
column = c.ToString
    Value = tb.text
    If c.DataType.ToString = "System.String" Then
     s = s + " @" + column + "='" + value + "', "
     Else
     s = s + " @" + column + "=" + value + ", "
    End If
   End IF
   FieldsCount = FieldsCount + 1
  Next c
  s = s + "@" + KeyField + "=NULL"
  me.Controls.Add(new LiteralControl(s))
  RunSql(s)
 End if
End Sub
Sub RunSql(vSql as string)
try
  Dim s As string
  Dim myConnection As OleDbConnection
  myConnection = New OleDbConnection(ConnStr)
  Dim mycommand As New OleDbCommand(vsql,myConnection)
  myconnection.Open()
  myCommand.ExecuteNonQuery()
  myconnection.Close()
  Catch ex As OleDbException
   ' SQL error
   Dim errItem As OleDbError
   Dim errString As String
   Dim s As string
   For Each errItem In ex.Errors
```

```
errString += ex.Message + "<br/>>"
```

Next

s = "
sql Error.Details follow:
" & errString

me.Controls.Add(new LiteralControl(s))

Catch myException as Exception

me.Controls.Add(new LiteralControl("Exception: " + myException.ToString()))

End try

End Sub

End Class

End Namespace

Using the GenEditAdd Custom Control

I will now show you how to hook up the GenEditAdd control to a DataGrid and use it to add, edit, and delete records in the masters table. The code for this discussion can be found in the ...GenEditAdd\GenEditAdd_final subfolder on the book's Web site at www.premierpressbooks.com/downloads.asp.

Each DataGrid that uses this control needs a "config" file. I have discussed the config_masters.aspx file earlier in this chapter and this is the config file used in this example. The DataGrid residing on the Web page "masters.aspx" requires two Hyperlink columns: one for the add mode and the other for the Edit mode. These Hyperlinks navigate to the config file and pass on a code_value of 0 in case of the add mode or a valid primary key in case of the edit mode as follows:

<property name="Columns">

<asp:HyperLinkColumn Text="Edit" DataNavigateUrlField="code_value"

DataNavigateUrlFormatString="config_masters.aspx?code_value={0}"/>

<asp:HyperLinkColumn Text="Add" DataNavigateUrlField="code_value"

DataNavigateUrlFormatString="config_masters.aspx?code_value=0"/>

</property>

The following is the complete code listing of Masters.aspx. Masters.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<%@ Register TagPrefix="Hersh" Namespace="Generic_chap7" Assembly="GenEditAdd_Chap7" %>

<%@Page Language="VB" %>

<html>

<script language="VB" runat="server">

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

if NOT (isPostBack)

rebind

end if

End Sub

Sub ReBind()

'DataSetCommand

SQL = "select m.*, g.code_display as category "

SQL = SQL + "from masters m, groups g "

SQL = SQL + " where m.code_category = g.code_value"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "masters")

'Binding a Grid

Grid1.DataSource=ds.Tables("masters").DefaultView

Grid1.DataBind()

End Sub

Sub RunSql(sql as string)

'Catch Control Validator errors

if not page.isvalid then

response.write("Stored Procedure did not execute")

rebind

exit sub

end if

try

Dim mycommand2 As New OleDbCommand(sql,myConnection)

myconnection.Open()

myCommand2.ExecuteNonQuery()

myconnection.Close()

'turn off editing

Grid1.EditItemIndex = -1

Catch ex As OleDbException

' SQL error

Dim errItem As OleDbError

Dim errString As String

For Each erritem in ex.Errors

errString += ex.Message + "
>"

Next

Response.write("SQL Error.Details follow:
br/>
" & errString)

Catch myException as Exception

Response.Write("Exception: " + myException.ToString())

End try

rebind

End Sub

Sub Grid1_delete(sender As Object, e As DataGridCommandEventArgs)

Dim code_value As string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString

Dim sql As string

sql = "Delete from masters where code_value = " + cstr(code_value)

RunSql(sql)

End Sub

</script>

<head>

<title>Masters DataGrid 1</title>

</head>

<body>

<form runat=server>

Chart of Accounts:

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

OnDeleteCommand = "Grid1_delete"

DataKeyField="code_value">

<Columns>

<asp:HyperLinkColumn Text="Edit" DataNavigateUrlField="code_value"

DataNavigateUrlFormatString="config_masters.aspx?code_value={0}"/>

<asp:HyperLinkColumn Text="Add" DataNavigateUrlField="code_value"

DataNavigateUrlFormatString="config_masters.aspx?code_value=0"/>

<asp:ButtonColumn Text="Delete" CommandName="Delete" HeaderText="Delete"/>

<asp:BoundColumn HeaderText="Account" DataField="code_display">

<HeaderStyle Width="150px"/>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Group" DataField="category">

<HeaderStyle Width="150px"/>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Type" DataField="type">

<HeaderStyle Width="150px"/>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Opening" DataField="opening">

<HeaderStyle Width="150px"/>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Closing" DataField="closing">

<HeaderStyle Width="150px"/>

</asp:BoundColumn>

</Columns>

<HeaderStyle BackColor="DarkRed" ForeColor="White" Font-Bold="true"/>

<ItemStyle ForeColor="DarkSlateBlue"/>

<AlternatingItemStyle BackColor="Beige"/>

</asp:DataGrid>

</form>

</body>

</html>

Summary

The GenEditAdd component that we built in this chapter allows us to provide editing and insert functionality to a DataGrid. The DataGrid does not have a built-in Insert mode and the GenEditAdd custom control fills this void. This control also replaces the Edit mode of the DataGrid (which was not automatic and required us to code a number of events). In the process of building this control I explained the theory underlying building custom controls. In <u>Project 3</u>, we will enhance this control to provide additional functionality as DropDown lists, required fields, read-only fields, and descriptive captions for column names.

Chapter 8: Business Objects

Business Objects are a library of functions and classes that can be used in any project. Commonly used code is encapsulated in a Business Object. This object serves as a "<u>service</u>" class to another object. It is instantiated as required and destroyed after use. A Business Object does not have any user interface.

The Bin Directory

If you have developed COM objects in the past you must know of the pain involved in registering components. A component had to be registered using the regsvr32.exe utility. If the component was modified, the entire Web server had to be stopped in order to reregister the component. ASP.NET has simplified the process of registering components. The components are simply copied and pasted in the bin directory. No registry updates are required, and to unregister you simply delete the component file from the bin directory. The original components may be replaced even when the Web server is running. ASP.NET allows all existing requests to complete and directs new requests to the new component.

Namespaces and Assemblies

Developers can group the internal code components (classes and interfaces) in namespaces. Such a logical organization also prevents collisions with classes written by other developers. A namespace provides a shortcut for referring to long class names. The Imports directive (Visual Basic.NET) and the using directive (in C#) are shortcuts that allow one to use namespaces without providing a fully qualified path.

The metadata for an object records information required to use the object. Typically, this information includes the name of the object, names of all the fields of the object, and their types and details of all member functions including parameter types and names.

An assembly allows for packaging applications into one comprehensive unit. Code compiled by the .NET compiler is converted to an intermediate form, called "IL." An assembly will contain all the IL, the metadata, and other required files.

Each assembly has a manifest, which contains information pertaining to the identity of the assembly (that is, name and version information) and other files contained in the assembly.

A Simple Business Object in Visual Basic

I will now build a simple object, which has one property and one method. The user sets the property message, and the method called test returns it to the calling form. The source code for this example can be found in the\basic\vb sub- folder on the book's Web site at www.premierpressbooks.com/downloads.asp.

- 1. Build the Component
 - 2. Imports System
 - 3. Imports System.Text
 - 4. Imports Microsoft.VisualBasic
 - 5. Namespace BasicObjVb
 - 6. Public Class BasicVb
 - 7. Private ls_message as string
 - 8. Public Sub New()
 - 9. MyBase.New()
 - 10. ls_message = ""
 - 11. End Sub
 - 12. Public Property message as string
 - 13. Get

14.	Return Is_message
15.	End Get
16.	Set
17.	ls_message = value
18.	End Set
19.	End Property
20.	Public Function test() as string
21.	Dim SB As StringBuilder
22.	SB = New StringBuilder(Is_message)
23.	SB = SB.Append("returned from function test")
24.	test = SB.ToString()
25.	End Function
26.	End Class
End N	amespace

This class is called BasicVb and it resides in the namespace BasicObjVb. It has one property called message, which, as usual, has a Set method, a Get method, and a local variable called ls_message. In a property syntax, the Get method is used to return the value stored in the local variable back to the calling object whereas the calling object writes to this property using the Set method.

The object has a Constructor event. This is the Sub new(). The Constructor event is fired as soon as the object is created. I have initialized the local variable ls_message to a single space in this event. Finally, I have a function called Test that returns a string. I use the StringBuilder class to build the returned string. This class has various methods for string manipulation. Some of its methods are Append, Replace, Remove, and ToString. I assign the message property to the StringBuilder and then use its Append method to add another string to it.

27. Compile the Object

This is done by running the following DOS command:

vbc /t:library /out:g:\aspnetsamples\bin\BasicObjVb.dll BasicObj.vb 28. **Build a Web Form**

Finally, you build a web form to test out the component. Figure 8.1 shows what the output looks like:



Figure 8.1: Simple Business Object. BasicObj.aspx

<%@ Import Namespace="BasicObjVb" %>

<html>

<script language="VB" runat="server">

Sub Page_Load(Sender As Object, E As EventArgs)

Dim s As string

Dim Comp As BasicVb

Comp = New BasicVb()

Comp.Message = "Hello World"

s = Comp.test()

response.write(s)

End Sub

</script>

</html>

In the web form, I import the BasicObjVb namespace with the import directive. I then declare a component of type BasicVb and assign Hello World to its message property. I call its test function, which returns the string that I store in a local variable called "s." Finally, I render this string to the browser using response.write.

A Simple Component in C#

I will now show you how to build the same object in C#. The source code for this example can be found in the\basic\c subfolder on the book's Web site at www.premierpressbooks.com/downloads.asp.

```
1. Build the Component
BasicObjC.cs
namespace BasicObjC {
 using System;
 using System.Text;
 public class BasicC{
  private String Is_message;
  public BasicC()
  ł
   //constructor
   ls_message = null;
  }
  public string message
  {
    get
    ł
     return ls_message;
   }
   set
   {
     ls_message = value;
   }
  }
```

```
public String test()
{
    StringBuilder SB = new StringBuilder(Is_message);
    SB.Append(" From test function...");
    return SB.ToString();
    }
    //class
    //namespace
```

Except for syntax changes, writing an object in C# is quite similar to writing it in Visual Basic. Note that the constructor in this case is the public method BasicC(). This is a public method with the same name as the class name. 2. **Compile the Object**

Run the bat file BasicObjC.bat which contains the following DOS commands:

csc /t:library /out:g:\aspnetsamples\bin\BasicObjC.dll BasicObjC.cs

This places the DLL file in the bin folder.

3. Build the Web Form

Finally, build an aspx form to test out the control.

BasicObjC.aspx <%@ Import Namespace="BasicObjC" %> <html> <script language="C#" runat="server"> public void Page_Load(Object sender, EventArgs E) {

```
BasicC comp = new BasicC();
```

comp.message = "Hello World";

```
String s = comp.test();
```

display.InnerHtml = s;

}

</script>

<h3>A Simple C# Component</h3>

<h5>Object Output: </h5>

```
<div id="display" runat="server"/>
```

</html>

Partitioning Services Between Web Forms and Components

The typical usage of a web form is to render HTML or XML pages to the browser. Since web forms reside on the server, can they be thought of as being business objects? In a component world, the answer would usually be negative. A web form contains both the presentation logic and business logic on the same page. A business object, on the other hand, is free from all presentation elements. It can be thought of as an object that provides *services* to another form. As a good *servant*, it is called whenever needed and dismissed when its need is over. This service analogy is also the reason that business objects are referred to as service classes. There are a number of advantages to using business objects. Some of these are summarized in the following:

 Promotes encapsulation: You encapsulate frequently used functionality into an object and expose just its properties and methods to the outside world.
 For example, most web forms require database access and manipulation routines. Instead of writing script in each page for database connection routines, we can create a business object that does this work for us. This object will expose properties and methods to the outside world and a user will not need to know its inner workings.

- Easy maintenance: Encapsulated code residing in one business object is far easier to maintain than code that is distributed over numerous Web pages. If our business logic changes, we just need to modify our business object and the consumers of the object need not do anything at all. This is analogous to cascading stylesheets. Suppose you want to change the definition of an H1 tag. You just do it in one place and the change is *cascaded* to all the pages that use that tag.
- Improves with reuse: One of the OO fundamentals is that codes reuse, refine, and improve the code. As more and more users test the object, the object is debugged thoroughly and increasing reliance can be placed on it. This promotes code reuse because developers can gradually build a tried and tested function library, which can then be shared with other developers.

A Database Class

I will now show you how to build a database class. This class will contain commonly used functionality for working with a database. It will include properties to set a database connection string and pass the object a SQL statement, which will be applied to the database. The SQL statement can return data (as in a SELECT statement) or perform action queries like update, insert, delete, or call a stored procedure, which returns no data. It will have two methods for applying the SQL statement to the database. The first method will return a DataView, which can be used to bind a control. The second method is a generic function to apply "action" SQL statements to the database. I will show you how to build the class, first in Visual Basic.NET, then in C#.

The Database Class in Visual Basic.NET

I start by building the component in Visual Basic.NET. The source code for this example can be found in the\SQLClass\ SqlClassvb subfolder on the book's Web site at www.premierpressbooks.com/downloads.asp.

1. The SQL Property: This is the SQL string that is passed by the user to this object. This can be any valid SQL statement. It is defined using the Set and Get methods and has an associated local variable

- ls_sql.
- 2. Private ls_sql as string
- 3. Public Property SQL as string
- 4. Get
- 5. Return ls_sql
- 6. End Get
- 7. Set
- 8. ls_sql = value
- 9. End Set

End Property

- 10. The ConnStr Property: This is the property that is passed the connection string by the user. It is implemented using the Set and Get methods and a local variable ls_connstr.
 - 11. Private Is_ConnStr as string
 - 12. Public Property ConnStr as string
 - 13. Get
 - 14. Return ls_ConnStr
 - 15. End Get
 - 16. Set
 - 17. ls_ConnStr = value

18. End Set

End Property

- 19. **Function Populate:** This function returns a DataView, which can be used to bind a control.
 - 20. Public Function Populate() As DataView
 - 21. Dim dv As DataView
 - 22. Dim i As integer
 - 23. Dim myConnection As OleDbConnection
 - 24. Dim myCommand As OleDbDataAdapter
 - 25. Dim ds As New DataSet
 - 26. myConnection = New OleDbConnection(ConnStr)
 - 27. myCommand = New OleDbDataAdapter(SQL, myConnection)
 - 28. myCommand.Fill(ds, "vTable")
 - 29. Populate = ds.Tables("vTable").DefaultView

End Function

This function uses the passed connection string to create a new OleDbConnection. The OleDbDataAdapter is used to populate a DataSet (and table vTable) using the passed SQL statement. The default view of table vTable is returned to the calling object.

- 30. Function RunSQL: We have met this function in earlier chapters. This is a generic function, which can be used to apply an "action" query to the database. Action queries are queries that do not return data. You can also use this function to run stored procedures that don't return data by calling them with the execute statement. For example, the statement Execute p_masters parameter a, parameter b, etc. will execute the stored procedure p_masters. The procedure p_masters could be a procedure that inserts a row in the Masters table.
 - 31. Function RunSql(vsql as string) as String
 - 32. Dim Message As string
 - 33. try
 - 34. Dim myConnection As OleDbConnection
 - 35. myConnection = New OleDbConnection(ConnStr)
 - 36. Dim mycommand As New OleDbCommand(vsql,myConnection)
 - 37. myconnection.Open()
 - 38. myCommand.ExecuteNonQuery()
 - 39. myconnection.Close()
 - 40. Catch ex As OleDbException
 - 41. Dim errItem As OleDbError
 - 42. Dim errString As String
 - 43. For Each errItem In ex.Errors
 - 44. errString += ex.Message + " "
 - 45. Next
 - 46. Message = "SQL Error.Details follow:
br/>
" & errString
 - 47. Catch myException as Exception
 - 48. message = "Exception: " + myException.ToString()

- 49. End try
- 50. RunSql = message
- End Function
- 51. **The Constructor Function:** The constructor function gets fired each time the object is initiated. I initialize the two properties to a space in this function.
 - 52. Public Sub New()
 - 53. MyBase.New()
 - 54. ls_sql = ""
 - 55. ls_ConnStr = ""

End Sub

The following is the complete listing of the database class:



Imports System.Data.OleDb

Imports System.Text

Namespace SQLNameSpace

Public Class SQLClass

Private Is_sql as string

Private Is_ConnStr as string

Public Sub New()

MyBase.New()

ls_sql = ""

ls_ConnStr = ""

End Sub

Public Property SQL as string

Get

Return Is_sql

End Get

Set

ls_sql = value

End Set

End Property Public Property ConnStr as string Get Return Is_ConnStr End Get Set ls ConnStr = value End Set End Property Public Function Populate() As DataView Dim dv as DataView Dim i As integer Dim myConnection As OleDbConnection Dim myCommand As OleDbDataAdapter Dim ds As New DataSet myConnection = New OleDbConnection(ConnStr) myCommand = New OleDbDataAdapter(SQL, myConnection) myCommand.Fill(ds, "vTable") Populate = ds.Tables("vTable").DefaultView End Function Public Function test() as string Dim SB As StringBuilder SB = New StringBuilder(ls_sql) SB = SB.Append("..returned from function test") test = SB.ToString() End Function Function RunSql(vsql as string) as String Dim Message As string try

Dim myConnection As OleDbConnection

myConnection = New OleDbConnection(ConnStr) Dim mycommand As New OleDbCommand(vsql,myConnection) myconnection.Open() myCommand.ExecuteNonQuery() myconnection.Close() Catch ex As OleDbException Dim errItem As OleDbError Dim errString As String For Each errItem In ex.Errors errString += ex.Message + " " Next Message = "SQL Error.Details follow:

/>/> & errString Catch myException as Exception message = "Exception: " + myException.ToString() End try RunSql = message End Function End Class

End Namespace

Compiling the Database Class

I have provided a bat file, which compiles the database class to a DLL. The following is the listing of the file SQLClass.bat:

set outdir=g:\aspnetsamples\bin\SQLClass.dll

set assemblies=System.dll,System.Web.dll,System.Data.dll,System.XML.dll

vbc /t:library /out:%outdir% /r:%assemblies% SQLClass.vb

pause

Running this file will compile and place SQLClass.dll in the bin folder.

Testing the Database Class

I have provided a web form, which tests the functionality of the database class. Figure 8.2 shows what it looks like.

A Dat	aBase Class in Visu:	al Basic				2
Retest	inset Update Delete	Deleted all to	at manda			
code y	value code display	code cat	eccry typ	e closir	accenina	
1	Cech in Hand	604	A	0	0	
2	Wells Fano Checkin	ng604	A	48	0	
3	Visa Card	604	A	40	0	
4	Salary	700	1	100	0	
5	Interest Income	700	i	0	0	
6	Rent	701	E	10	0	
7	Utilities	701	E	0	0	
8	Medical Expenses	701	E	0	0	
10	Entertainment	704	E	2	0	
10	Entertainment	/04	E	4	0	

Figure 8.2: Testing the database class.

The following is the code listing: TestVbClass.aspx

<%@ Import Namespace="SQLNameSpace" %>

<html>

```
<script language="VB" runat="server">
```

Dim Comp As SQLClass

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

Comp = New SQLClass()

```
Comp.ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"
```

if NOT (isPostBack)

rebind

end if

End Sub

Sub Show_Click(Sender As Object, E As EventArgs)

Message.Text = "Masters Table Displayed... "

ReBind

End Sub

Sub Insert_click(Sender As Object, E As EventArgs)

```
sql = "Insert into Masters(code_display,code_category,type)"
```

```
sql = sql + "Values ('test',701,'E')"
```

Comp.RunSql(sql)

rebind

Message.Text = "Inserted test record... "

End Sub

Sub Delete_click(Sender As Object, E As EventArgs)

sql = "delete from masters where code_display = 'test'"

Comp.RunSql(sql)

rebind

Message.Text = "Deleted all test records..."

End Sub

Sub Update_Click(Sender As Object, E As EventArgs)

sql = "UPDATE Masters Set Opening = 90 WHE RE code_display = 'test'"

Comp.RunSQL(sql)

rebind

```
Message.Text = "Updated all test records: Set closing balance = 90...! "
```

End Sub

Sub ReBind()

```
Comp.SQL = "select * from Masters"
```

DataGrid1.DataSource=Comp.populate()

DataGrid1.DataBind()

End Sub

</script>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<h3>

A DataBase Class in Visual Basic

</h3>

<form runat="server">

<asp:button text="Refresh" Onclick="Show_Click" runat="server" />

<asp:button text="Insert" Onclick="Insert_Click" runat="server" />

<asp:button text="Update" Onclick="Update_Click" runat="server" />

<asp:button text="Delete" Onclick="delete_Click" runat="server" />

<asp:label id="Message" runat="server" />

<asp:DataGrid id="DataGrid1" runat="server" />

</form>

</body>

</html>

- 1. **Initiating the Object:** The object is initiated in the page_load event of the web form and the connection string passed to it is as follows:
 - 2. Sub Page_Load(Source As Object, E As EventArgs)
 - 3. Comp = New SQLClass()
 - 4. Comp.ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;"
 - 5. if NOT (isPostBack)
 - 6. rebind
 - 7. end if
- End Sub
- 8. The ReBind function passes a SQL query to the object and binds a DataGrid to the DataView returned by the object as follows:
 - 9. Sub ReBind()
 - 10. Comp.SQL = "select * from Masters"
 - 11. DataGrid1.DataSource=Comp.populate()
 - 12. DataGrid1.DataBind()

End Sub

13. The Insert_click event executes an insert statement against the database using the RunSQL function of the object.

- 14. Sub Insert_click(Sender As Object, E As EventArgs)
- 15. sql = "Insert into Masters(code_display,code_category,type)"
- 16. sql = sql + "Values ('test',701,'E')"
- 17. Comp.RunSql(sql)
- 18. ReBind
- 19. Message.Text = "Inserted test record... "

End Sub

- 20. The Update_Click event executes an update statement against the database using the RunSQL function.
 - 21. Sub Update_Click(Sender As Object, E As EventArgs)
 - 22. sql = "UPDATE Masters Set Opening = 90 WHERE code_display = 'test'"
 - 23. Comp.RunSQL(sql)
 - 24. ReBind

25. Message.Text = "Updated all test records: Set closing balance = 90...! "

```
End Sub
```

```
26. The Delete_click event executes a delete statement using the RunSQL function.
```

- 27. Sub Delete_click(Sender As Object, E As EventArgs)
- 28. sql = "delete from masters where code_display = 'test'"
- 29. Comp.RunSql(sql)
- 30. ReBind
- 31. Message.Text = "Deleted all test records..."

End Sub

The Database Class in C#

In the following section, I include a complete listing of the database class written in C#. Except for syntax changes, this class is similar to the Visual Basic.NET class discussed in the preceding section, hence I have not gone into a detailed discussion of the code. The source code for this example can be found in the\SqlClass\ SqlClassC subfolder on the book's Web site at <u>www.premierpressbooks.com/downloads.asp</u>.

SQLClassC.cs

namespace SQLNameSpaceC

{

using System;

using System.Data;

using System.Data.OleDb;

using System.Text;

public class SQLClassC

{

private String Is_connStr;

private String Is_sql;

public String ConnStr

```
{
```

get

{

return ls_connStr;

```
}
```

```
set
 {
  ls_connStr = value;
}
}
public String SQL
{
 get
 {
  return ls_sql;
 }
 set
 {
  ls_sql = value;
}
}
public DataView Populate()
{
 //for queries that return data
 //and for binding controls
 OleDbConnection myConnection = new OleDbConnection(ConnStr);
 OleDbDataAdapter myCommand = new OleDbDataAdapter(SQL, myConnection);
 DataSet ds = new DataSet();
 myCommand.Fill(ds, "vTable");
 return ds.Tables["vTable"].DefaultView;
}
```

```
public String RunSQL( string vsql)
```

```
{
 try
 {
  OleDbConnection myConnection = new OleDbConnection(ConnStr);
  OleDbCommand mycommand = new OleDbCommand(vsql, myConnection);
  myConnection.Open();
  mycommand.ExecuteNonQuery();
  myConnection.Close();
 }
 catch(Exception e)
 {
  string ret = "Exception: " + e.ToString() ;
}
 return("OK");
}
public String test()
{
 StringBuilder SB = new StringBuilder(ls_connStr);
 SB.Append(" ," + ls_sql);
 return SB.ToString();
}
```

Compiling the C# Class

}

The database class is compiled into a dll and placed in the bin folder, as in the following: **SQLClassC.bat**

set outdir=g:\aspnetsamples\bin\SqlClassC.dll

set assemblies=System.dll,System.data.dll

csc /t:library /out:%outdir% /r:%assemblies% SqlClassC.cs

```
pause
```

Testing the C# Class

This web form tests the Populate and RunSQL methods of the C# database class. TestcClass.aspx

```
<%@ Import Namespace="SQLNameSpaceC" %>
```

<html>

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

```
SQLClassC comp = new SQLClassC();
```

```
public void Page_Load(Object sender, EventArgs E)
```

{

```
comp.SQL = "Select * From Masters";
```

```
comp.ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASPNET;User ID=sa;";
```

bind();

```
string sql = "Insert into Masters(code_display,code_category,type)";
```

```
sql = sql + "Values ('test',701,'E')";
```

comp.RunSQL(sql);

}

```
public void Insert_Click(Object sender, EventArgs E)
```

```
{
```

```
string sql = "Insert into Masters(code_display,code_category,type)";
```

```
sql = sql + "Values ('test',701,'E')";
```

```
comp.RunSQL(sql);
```

bind();

}

```
public void Update_Click(Object sender, EventArgs E)
  {
   string sql = "UPDATE Masters Set Opening = 90 WHERE code_display = 'test'";
   comp.RunSQL(sql);
   bind();
  }
  public void Delete_Click(Object sender, EventArgs E)
   {
   string sql = "delete from masters where code_display = 'test'";
   comp.RunSQL(sql);
   bind();
  }
  public void bind()
  {
   DataGrid1.DataSource=comp.Populate();
   DataGrid1.DataBind();
  }
 </script>
 <body>
  <h3><font face="Verdana">SQL Class in C# </font></h3>
  <form runat=server>
   <asp:button text="Insert" onclick ="Insert_Click" runat=server/>
   <asp:button text="Update" Onclick="Update_Click" runat=server/>
   <asp:button text="Delete" Onclick="Delete_Click" runat=server/>
   <asp:DataGrid id="DataGrid1" runat="server" />
  </form>
 </body>
</html>
```

Summary

Developing business objects is an important topic. Well-encapsulated code is easy to maintain, refines with reuse, and can be shared easily among developers. It is a great way of separating presentation from code.

Database manipulation functions are a prime candidate for encapsulation in Business Objects. This chapter developed generic functions to apply SQL statements and queries to the database. These functions were capable of returning data, applying "action" queries such as insert, update, and delete, and executing stored procedures.

Chapter 9: Working with ASP.NET Web Services

Overview

Developing applications has come to mean developing for the Web. Developing server applications that run on any operating system that can host HTTP applications has become the goal. The two technologies that have come to the forefront are SOAP and XML. Functionality residing in a Business Object can be accessed using the SOAP protocol and exchanged as XML data and the technology that makes this happen is web services.

The simplest way to describe <u>web services</u> is that it is a library of useful functions, encapsulated within a Business Object that is accessed using SOAP and XML. Actually, we can also use HTTP Get and Post to transport the data. However, Get and Post are limited to sending and receiving name/value pairs of data whereas we can use SOAP to serialize complex structures, such as ASP.NET DataSets, complex arrays, custom types, and XML nodes. SOAP is a standard for encoding inter-machine function calls in XML. Exchanging information as XML enables a client application to call a function on the server, regardless of what operating system each is running, what programming language each is written in, or what component model is supported on each. This is because XML is basically a text file, which all machines can understand, and because SOAP uses HTTP, which is also the most common Internet transfer protocol and one that is used by essentially all Web browsers.

Application-to-application communication is not a new idea. Technologies, such as DCOM, RPC, and MSMQ (*Microsoft Message Queue Service*) already exist to enable this type of communication. The major limitation of these technologies is that it works from one similar system to another. A DCOM system works well for Microsoft-only systems and requires a DCOM client and a DCOM server. These technologies require one to buy into and commit to a particular architecture. We cannot have the situation where any Internet client makes a function call on any server. Another limitation of these technologies is that they have problems communicating over corporate firewalls. SOAP uses HTTP and can pass through firewalls.

Web services are an easy concept to understand. A Business Object is called a service. Such an object is a non-visual object that contains a number of functions, grouped together according to some logical classification. It is called a "<u>service</u>" because it seeks to "serve" another object, which can be an HTML form, a web form, or any other objects that can "initiate" it. As a good servant, it stays around for as long as the calling object requires it and is then dismissed. In ASP.NET you write a web service as you would write a normal Business Object and the only difference is that the functions are preceded with a special <WebMethod() > attribute that marks them as web services.

Web services may be third-party application libraries that can be invoked over the Internet. Examples of web services might include shipment tracking (similar to the functionality provided by the FedEx site), credit card verification, and a spell checker.

Web services brings some exciting possibilities to the traditional way of designing applications. Using ASP.NET and web services, we can design applications that can send and receive data using Internet and HTTP. Consider an accounting application.

Traditionally, accounting applications have been developed around the client/server model. The modules (Financial Accounting, Inventory, Invoicing, and so on) share a central database and are connected to each other with some sort of a network protocol. Using web services, the various modules of an accounting application need no longer be connected with "wire."

Take the example of a manufacturing company that has its inventory location in one state and its finance department in another. The financial and inventory records need to be integrated, so a central database is used. In a traditional client/server model, the financial accounting software, inventory software, and the central database would be hooked together by some sort of satellite link. In our web services model, however, we use Internet (HTTP) as our link. Data would be exchanged over the Internet (HTTP) using XML and the SOAP protocol. Both the modules could be written in different programming languages and could work off different operating systems. Because they talk SOAP and XML, they can both access a central repository of functions, which interacts with the database.

Writing a Simple Web Service

To demonstrate the techniques involved in creating web services, I have built a web service, which has two simple functions called TestFunction() and add(). The function TestFunction() returns a message string based on the Boolean variable passed to it. The function add() returns the addition of two integers passed to it. To build the service, I have used a simple text editor and saved the file with an .asmx extension. The source file is called BasicService.asmx and can be found in the ... \Basic\GetPost subfolder of the samples folder for this chapter on the book's Web site.

BasicService.asmx

<%@ WebService Language="VB" Class="TestService"%>

Imports System

Imports System.Web.Services

Public Class TestService: Inherits WebService

<WebMethod()> Public Function TestFunction (vInput as Boolean) As String

If (vInput = TRUE) Then

TestFunction = "It is the truth..."

Else

TestFunction = "False!False!False"

End if

End Function

<WebMethod()> Public Function add(a as integer, b as integer) as string

add = cstr(a+b)

End function

End Class

The WebService attribute informs ASP.NET to expose the code as a web service and the language directive chooses Visual Basic.NET as our compiling language. The class attribute specifies the class, which will handle the incoming function calls. In this case it is the TestService class. The System and System. Web namespaces contain prefabricated features that I need in the web service, so I import them using the Imports directive. The TestService class inherits from the WebService class. An inherited class can override and/or extend functionality of its parent class. When *importing* a namespace, you cannot extend the functionality of the imported class, whereas with *inheriting*, you can extend the functionality.

I now add functions to my new class. I have two functions in the TestService class. These are the functions TestFunction and Add. The process of defining functions is no different from the normal Visual Basic function definition syntax. However, you add a new attribute <WebMethod() > to methods that you want to expose as web services. ASP.NET exposes all methods with this attribute as web services.

Testing the Service

ASP.NET contains prefabricated support for testing the functionality of the web service. If I open the TestService.asmx file in my browser, through IIS (that is, a fully qualified URL that goes through a localhost), I see the page shown in <u>Figure 9.1</u>. This page reads the asmx file and presents a way to test all the functions marked as WebMethods. I can test the TestFunction method or the add method by supplying the requisite parameters. The result of the function call is displayed as XML via the GET protocol.



Figure 9.1: Testing the service.

The WSDL Contract

The Web Services Description Language (or WSDL in short) contract is an XML-based file, which fully describes a web service. It is similar to the type library of a COM object. A type library contains information pertaining to the component's unique identifier (the CLSID), the interfaces implemented by the component, and the method signature of each interface. In a similar manner the WSDL file (or contract) is a file that shows all the methods contained in a web service, the parameters they expect, and the protocols supported. A type library can only be used with Microsoft systems since it is Microsoft specific whereas the WSDL contract being an XML file, can be used with non-Microsoft systems.

This file is a "contract" with the outside world, which specifies what the web service is capable of doing. When you view the TestService.asmx file through IIS, you can click on the hyperlink Service Description and you will view the XML file, which contains

the contract. You can also obtain the contract by appending the fully qualified URL of the asmx file with a <code>?WSDL</code> string. For example,

http://localhost/test/TestService.asmx?WSDL. The WSDL file specifies what protocols are available to a calling client and provides information regarding how the function call should be made. The WSDL file for the methods <code>TestFunction</code> and Add has sections for accessing the Web Service using the SOAP, HTTP Post, or HTTP Get protocols. Further, it tells us how we should make the call and what parameters to pass to the functions.

Though a detailed knowledge of the WSDL contract is not required, I analyze its important element tags below.

<operation>

Each method of the web service is operation of the web service. Thus methods like the Add and TestFunction are both operations. According to the WSDL specification "Operation is an abstract description of an action supported by the service."

<definitions>

The <definitions> element is the root element of the WSDL document. All other elements reside within its tag boundaries. Its attributes define the target namespace and other namespace definitions.

<service >

The <service > element tag states the name of the web service that the WSDL contract describes. The web service that I have just created resides in the TestService class and the <name> attribute for this web service is this class name. This is evident from the following extract of the WSDL contract.

<service name="TestService">

</service>

<port>

Within the <<u>service</u> > element tag, the WSDL contract specifies the different ports on which the web service is accessible. The address of the web service (for each protocol) is provided here as a fully qualified URL. The location of the web service on my local machine is provided as:

http://localhost/AspNetSamples/9_Samples/Basic/GetPost/BasicService.asmx

The relevant extract from the WSDL contract for the SOAP protocol is as follows:

<service name="TestService">

<port name="TestServiceSoap" binding="s0:TestServiceSoap">

<soap:address location="http://localhost/AspNetSamples/

9_Samples/Basic/GetPost/BasicService.asmx" />

</port>

</service>

You can make the web service accessible through multiple ports. This web service is accessible through SOAP, HTTP GET, and HTTP Post. You can note from the WSDL contract that the <http://ddress/>element tag is quite similar to the

<soap:address> element tag listed above. On my machine it is as follows:

<http://localhost/AspNetSamples/

9_Samples/Basic/GetPost/BasicService.asmx" />

<message>

Communication with a web service involves making a request (a function call) and getting a response back from the web service. Consequently, you will note that each function described in the WSDL contract has an In (request) and an Out (response) message element tag for each protocol. For example, there are a total of six In/Out element tags for the Add method (one pair each for the SOAP, HTTP GET, and HTTP Post).

The <part> element tag corresponds to the parameter or return value from the remote procedure call. The following extract shows the parameters expected by the Add method when using the HTTP GET protocol.

<message name="addHttpGetIn">

<part name="a" type="s:string" />

<part name="b" type="s:string" />

</message>

<message name="addHttpGetOut">

<part name="Body" element="s0:string" />

</message>

The <message> element tag needs the <operation> element tag to actually identify the input and output messages as follows:

<operation name="add">

<input message="s0:addHttpGetIn" />

<output message="s0:addHttpGetOut" />

</operation>

<binding>

Finally the WSDL contract needs to specify how the web service expects the data to be encoded. For example the TestFunction requires a Boolean parameter (vInput) which can be represented as -1, 1 or True. The <binding> element tag specifies that we will be abiding by the SOAP specification which contains predefined rules for such situations as follows:

<binding name="TestServiceSoap" type="s0:TestServiceSoap">

<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />

</binding>

Calling the Service Using HTTP Get

In this section, I will make a call to the web service using the HTTP Get protocol. Let's take a look at the WSDL portion that describes the HTTP Get protocol for the function TestFunction. The relevant extract is as follows:

```
WSDL extract for HTTP Get
```

<message name="TestFunctionHttpGetIn">

```
<part name="vInput" type="s:string" />
```

</message>

<message name="TestFunctionHttpGetOut">

<part name="Body" element="s0:string" />

</message>

This tells us that the test function requires a single parameter value called vInput.

The request is to be made as a string and the response will be returned as XML. Appending a question mark, the function name, together with the required parameters after the fully qualified name of the web service form will make the request. This call is made as follows:

http://localhost/path name/basicservice.asmx/TestFunction?vInput=TRUE

The source code for this example (basicHTTPGet.html) can be found in the ...\basic\GetPost subfolder of the samples folder for this chapter on the book's Web site. Please note that all the file location paths should be modified to point to your application folder. Figure 9.2 shows a sample call to the TestFunction method.

Screening Surveyer Spectral systems and Spin cabarat Spage are	6) sarpes
Hill R. Hudsch 296ch, ng/Capir-Man/Self-right Hud	
ITP-GET Example	
<u>6. Connect de la COME</u> Rémonstrieur d'Anne Rémonstrieur Phin	

Figure 9.2: A call to the web service using the HTTP Get protocol. basicHTTPGet.html

<html >

<head>

</head>

<body>

<H4>HTTP GET Example</H4>

<a href="http://localhost/ASPNETSamples/

9_Samples/basic/GetPost/basicservice.asmx">

WSDL Contract

<a href="http://localhost/ASPNETSamples/9_Samples/basic/

GetPost/basicservice.asmx/TestFunction?vInput=TRUE">

TestFunction?vInput=TRUE

<a href="http://localhost/ASPNETSamples/9_Samples/basic/

GetPost/basicservice.asmx/TestFunction?vInput=FALSE">

TestFunction?vInput=False

</body>

</html>

Clicking on any of the function links calls the TestFunction and passes to it the appropriate input parameter. For example, clicking on the vInput = "True" link passes true to the TestFunction. The web service returns the response in XML as follows:

<?xml version="1.0" encoding="utf-8" ?>

<string xmlns="http://tempuri.org/">It is the truth...</string>

Calling the Service Using HTTP Post

In this section, I will make a call to the web service using the HTTP Post protocol. Let's take a look at the WSDL portion that describes the HTTP Post protocol for the function TestFunction. The relevant extract is as follows:

```
WSDL extract for HTTP Post
```

<message name="TestFunctionHttpPostIn">

<part name="vInput" type="s:string" />

</message>

<message name="TestFunctionHttpPostOut">

<part name="Body" element="s0:string" />

</message>

In order to call the TestFunction using the HTTP Post protocol, I need to pass the parameter vInput to the web service via an input control residing within form tags. A sample web form that makes this request is shown in <u>Figure 9.3</u> and its listing is shown below. Please note that the <action> attribute of the <form> tag should be modified to point to your application folder. The source code can be found on the ... \Basic\GetPost subfolder of the samples folder for this chapter at www.premierpressbooks.com/downloads.asp.



Figure 9.3: A call to the web service using the HTTP Post protocol. basicHTTPPost.html

<html>

<head>

</head>

<body>

```
<h1>HTTP Post Example</h1>
```

<hr>

<form METHOD="POST" ACTION="http://localhost/ASPNETSamples/

9_Samples/basic/GetPost/basicservice.asmx/TestFunction">

<blockquote>

```
<input TYPE="RADIO" NAME="vInput" VALUE="True" CHECKED>
```

```
True <input TYPE="RADIO" NAME="vInput" VALUE="False">
```

False


```
</blockquote>
```

```
<input TYPE="SUBMIT" VALUE="Submit Form">
```

</form>

<hr>

</body>

</html>

Calling Services Using SOAP

SOAP is the most important transport protocol. You will note above that the Get and Post protocols are limited to sending and receiving name/value pairs of data. SOAP is important because we can serialize complex structures like DataSets, complex arrays, custom types, and XML nodes with it. A client makes a SOAP call by sending a message encoded in SOAP's XML vocabulary and transmits it to the server over HTTP. This message has an outer envelope, within which there is an optional header and a required body. The body contains the function name and the parameters that are to be passed to the function. Each of these elements is prefixed with an XML namespace (http://schemas.xmlsoap.org/soap/envelope).

A SOAP request for the TestFunction could be as follows:

<?xml version="1.0"?>

<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope"

SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">

<SOAP:Body>

<TestFunction>

<vInput>True</vInput>

</TestFunction>

</SOAP:Body>

</SOAP:Envelope>

The SOAP request is "posted" to a "SOAP Listener" on the server. The listener is typically the web services form. The communication between the client and server is by HTTP, though in theory you could use transfer mechanisms like sockets, message queuing, or e-mail. The SOAP Listener executes the function called and returns the response in XML. A successful response is enclosed within an element tag, which has the name of the original method and a suffix of response. Thus, the response to the TestFunction would be enclosed within an element tag TestFunctionResponse. The SOAP Body of the response could look like the following:

<?xml version="1.0"?>

<SOAP:Envelope xmlns:SOAP=http://schemas.xmlsoap.org/soap/envelope

SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">

<SOAP:Body>

<TestFunctionResponse>

<return>it is the truth</return>

</TestFunctionResponse>

</SOAP:Body>

</SOAP:Envelope>

Luckily, we do not have to go into all the intricacies of setting up infrastructures of communicating using SOAP and XML. ASP.NET takes care of all that. You generate a proxy of your web services form using the wsdl.exe utility provided by ASP.NET. This tool reads the WSDL file and generates a proxy using the language of your choice. Whenever the client makes a call to a function, the proxy generates an HTTP request and sends it to the server. When the response is received, the proxy parses and returns the result. I will now show you how to implement the SOAP protocol in ASP.NET. I will follow a step-by-step approach. The sample files for this example can be found in the .../basic/soap subfolder of the samples folder on the book's Web site at www.premierpressbooks.com/downloads.asp.

- 1. Create the Web Service file. I have already created the BasicService.asmx file, so I will just copy it to the folder.
- Create the WSDL file. Open BasicService.asmx so that it goes through IIS (such as http://localhost/your virtual directory/BasicService.asmx). Click on the Service Description hyperlink. Save the resultant file as basicService.wsdl. Note that you can also do the same thing by browsing to http://localhost/your virtual directory/BasicService.asmx?wsdl.

Caution

You do have to re-create this WSDL file even though there is a WSDL file in the sample folder. This file is specific to the application folder on my machine and you have to replace this file with your application folder-specific WSDL file as described above.

3. Run mbasicService.bat. This bat file does two tasks. First, it makes a proxy using the command-line utility wsdl.exe, which comes with the .NET SDK. It reads the description of the web services from the WSDL file and creates a proxy class with the extension of .vb (since we are using Visual Basic). This class has the class name specified in the asmx file. Thus, my proxy class is called Testservice.vb. I then compile the proxy class and put the resultant DLL (BasicService.dll) in the bin folder. If everything goes well, you should have two new files created. The proxy file TestService.vb (in the local folder) and the BasicService.dll (in the bin folder). Visual Studio automates this task of creating a proxy, and later in this chapter I shall be looking at creating a web service using Visual Studio. Figure 9.4 shows the output from the console at this stage.



mbasicService.bat

REM -----Make Proxy------

wsdl.exe /I:VB /n:NameSpaceHersh /out:TestService.vb BasicService.Wsdl

REM ----- Compile Proxy------

Rem Remember to change outdir variable to point to your bin folder

set outdir=g:\AspNetSamples\bin\BasicService.dll

set assemblies=System.dll,System.Web.dll,System.Data.dll,

System.Web.Services.dll,System.Xml.dll

vbc /t:library /out:%outdir% /r:%assemblies% TestService.vb

pause

4. Test the service. I have created a simple web form to test the service. This form calls the TestFunction and passes the parameter true to the web service. The return value from the function is written out to the screen. This is shown in <u>Figure 9.5</u>.



<html>

```
<script language="VB" runat="server">
  Protected Sub Page_Load(Src As Object, E As EventArgs)
   Dim t As New NameSpaceHersh.Testservice
   Dim s As string
   s = t.testfunction(true)
   message.text = "Return from function : " + s
  End Sub
 </script>
 <body style="font: 10pt verdana; background-color:beige">
  <center>
   <h2><b>Testing Soap Proxy</b></h2>
   <form runat="server">
    <asp:Label id="Message" runat="server"/>
   </form>
  </center>
 </body>
</html>
```

Creating a Web Service Using Visual Studio

Building a web service with Visual Studio.NET (VS .NET in short) involves creating a new web services project and adding methods and properties to a web services class form (i.e. an asmx form). This section builds a web service with Visual Studio.NET and demonstrates the use of the integrated VS debugger. A subsequent section will build a Visual Studio.NET project to consume the web service developed here.

You will find the source code for this example in the VSService subfolder of the samples folder for this chapter. To install the sample files on your machine, create a new web service project and import the sample files into the new project. This will create the required application virtual folder on your machine.

- 1. Start Visual Studio.NET.
- 2. Select File/New/Project. Figure 9.6 shows VS.NET at this stage.

E Hereinen Derertegenund	Instrument [drags] - Uart Page			
See See	tendur Dela → Ja (post OdrSaltet → Sh Odr4 → BertSaldon	3 ∰ Litelydecentr 3 = 1 + #	12983	· O .
Apit-logic A	Bert SAGA.	Anne Hervel Hervel Mediaeriadur CPrintiya Dipene Fregerit New Progerit Report a Vayori Dashe B(Trosa	6 (16-05-6 16-05-00-18 40-70-00-18 40-7	
11 ongos 🕞 (11 ongos) Incos		1	1 1	

Figure 9.6: New project.

3. Select Visual Basic Projects from the left pane and Web Service from the right. Figure 9.7 shows VS.NET at this stage.

ew Project				22	122
Visual Bas Visual C# Visual C# Visual C# Visual For Setup any Other Pro Visual Stu	aic Projects Projects + Projects & Projects d Deployment Projects ojects dio Solutions	Windows Application Web Application	Cass Lbrary	Windows Control Library Web Control Library	
A project for crea	ting Web services to use from	m other applications	p.		
Name:	VSService				
ocation:	http://localhost/aspbc	ok_wip/Chapter9/VS/	WebService 💌	Browse	
Project will be crea	ated at http://localhost/aspbo	xok_wip/Chapter9/VS	WebService/VSS	iervice.	
¥ Morg	J [ОК	Cancel	Help	

Figure 9.7: New Web Service.

4. Type in VSService for the name. You might get a Web Access Failed dialog box as shown in Figure 9.8.

Web Access Failed	×
The default web access mode for this project is set to FrontPage, bu http://localhost/aspbook_wep/Chapter9/VSWebService/VSService/ returned was:	t the project folder at most be opened with FrontPage. The error
Unable to create Web project 'aspbook_wip/Chapter9/VSWebService folder "c:\inetpub\www.root\aspbook_wip\Chapter9\VSWebService\V	WSService', Server error: Cannot create SService'',
What would you like to do?	
Try to open the project with a file share path:	
Location: c:\inetpubl\www.root\aspbook_wip\Chapter9\V5WebS	Service VSService
C Workeffine	
	X Cancel Help

Figure 9.8: Web Access Failed dialog box.

Click on OK and Visual Studio.NET will then use file access to open the solution.

5. Visual Studio.NET generates a new Solution, which creates a references folder and four files. The web.config is an XML file, which contains various configuration options (for example, session timeout interval), and

which controls the web service at runtime. The file .disco is an XML file used for dynamic discovery of web services by clients. This means that when we want to use this service in another project, we navigate to this file and VS.NET adds a reference to the service for us. The file globals.asax is where project-wide event handlers (like

ApplicationStart and ApplicationEnd) reside. Figure 9.9 displays the four files in the solution explorer.

Solu	tion E	xplorer - VSService	
٢		B	
-	Soluti	on 'VSService' (1 project)	

ė 🚱	VSService
÷	💿 References
	📑 Config.web
	🛐 Global.asax
	🚳 Service1.asmx
	🗑 VSService.disco

Figure 9.9: The Solution pane displaying the four default files.

- 6. Right-click on Service1.asmx and select Open. If you open this file in Notepad, you will note that this file makes a callout to code that resides in another file (Service1.vb) as follows:
 - 7. <%@ WebService Language="vb" Codebehind="Service1.asmx.vb"
 - Class="VSService.Service1" %>

Add the two functions ${\tt TestFunction()}$ and ${\tt add()}$ within the class module as follows:

<WebMethod()> Public Function TestFunction (vInput as Boolean) As String

If (vInput = TRUE) Then

TestFunction = "It is the truth..."

Else

8.

TestFunction = "False!False!False"

End if

End Function

<WebMethod()> Public Function add(a as integer, b as integer) as string

add = cstr(a+b)

End Function

Figure 9.10 displays VS.NET in the code view at this stage.



Figure 9.10: Creating the web service.

9. Right-click on the Solution VSService and choose Build as shown in



Figure 9.11: Building the solution.

10. Test the service by right-clicking on Service1.asmx and selecting View in Browser. You will see the default test page as shown in Figure 9.12 and can test the functions by supplying the required parameters.

File 64 Mere Farceles Tauls Help	
0 0 0 Elevante Quest Elevante 0 0 - 8 E - 9 0	
Alter Dep de des primer de set anne	. 24
an Dissertan Disertand Clark stops should Observation Character	
Service1	
The following operations are supported. For a formal definition, please review the <u>Service Description</u> .	
• and	
Testfunding	
This web service is using http://temport.org/ as its default namespace.	_
Recommendation: Change the default namespace before the web service is made public.	
tack web service rends a unique nomempage to identify it as that client applications can distribuid in them other services on the web. http://tempusi.org/is a web services that are under devolupment, but published web terretes chould use a more permanent namespace.	reliable for
theor was service shauld be identified by a namespace that you control, her maniple, you multi-us your company's baseter domain name as part of the nam Attrough many web convice namespaces lock like URLs, they need not point to an extral resource on the web. (Web convice namespaces are URLs.)	wapase.
For ASP 301 Web Services, the default remempance can be charged using the WebService attribute's tensorpace property. The WebService attribute is an ethic to the address that and uses the web service methods. Being in a code example that sets the superspace to 2My (Journault com/antiservices/)	Bute appied
CF	
[Meddacvice:Chemospher=Toty://wilscondt.com/webmervices/")] public class ByDedacvice (// Implementation,)	
Vesal Inect NC7	
cHeldervice(hereparce*hitp://wincomft.com/wineevices/*j>:Fublic Class Byhithevice * oppresentation Fublic Class Byhithevice	-
For more datafs on XM, namespaces, see the W3C recommendation on Hammanation in 2010.	
For more details as write the detail in an detail in a detail of a	أد ال

Figure 9.12: Testing the functions.

11. Visual Studio.NET includes a powerful debugger. You can set breakpoints and step through code by pressing F8. To use the debugger, open Service1.asmx and set a breakpoint by double-clicking on the left side (gray) pane, outside the code area. A red dot will appear at the point where the pane is clicked. This is shown in <u>Figure 9.13</u>.



Figure 9.13: Setting a breakpoint.

Start the Debug mode by pressing F5. If prompted for a debug folder, choose any folder not in wwwroot (for example c:\test). The default test page will now be presented. Supply the parameters for the function being debugged. The debugger will bring you back to the breakpoint line. If you bring your mouse on a parameter and leave for a second, you will be able to see the parameter passed as shown in Figure 9.14. Step through the code by pressing F8.



Figure 9.14: Using the debugger.

12. Browse to the bin folder in VSService folder. Note that a compiled DLL called VSService.dll already exists in this folder. Visual Studio.NET created the proxy and compiled it for us.

Calling the Web Service from a Web Form

In this section, I will build a client application, which will use the web service developed in the preceding section. This application is a VS Web Application and is created as explained in the following steps.

- 1. Start Visual Studio.NET.
- 2. Select File/New/Project.

3. Select Visual Basic Projects from the left pane and ASP.NET Web Application from the right as shown in Figure 9.15. Type in VSClient for the project name.

	Templates:		88 2
c Projects Projects Projects Deployment Projects ects So Solutions	Windows Application ASP.NET Web Argitation	Class Library Class Library ASP.NET Web Service	Windows Control Library Web Control Library
VSClient			
http://localhost		¥	Browse
	c Projects Projects Projects Deployment Projects ects to Solutions Solutions	Templates: Projects Projects Deployment Projects ects to Solutions Projects Deployment Projects ects Solutions Migna an application with a Web user interface VSCIent Intp://localhost	Templates: Projects Projects Projects Deployment Projects ects to Solutions Mindows Application Application ASP_INET Web Service Interface VSCIlent Intp://localhost

Figure 9.15: New Project pane.

4. Drag three textboxes, three labels, and one button onto the form. Rightclick on an empty portion of the form and select Properties. Make sure that the Page Layout property is GridLayout. With GridLayout you are able to drag the controls and position them visually on the form. Change the Text properties of three labels to read Parameter A and Parameter B and Result, respectively, and ID properties of the three textboxes to read Param1 and Param2, and ResultAdd respectively. This form should look like the one displayed in Figure 9.16.

To Walkest - Novemb Wand Description	ET [de sign] - Webstares Longe			
the Lik yes from build the	Debug Parriel Table Street Planes 3	inte Mindon Dela		
10-0-000 2 40	B an a con all - En a bebag	a ga Advit Arcentor	· 3823-0	
1-10 + 1 P .1	¥[-]	A H / H / H	ABBBBBCCC	F (F .)
Andres A & A	statement and state			Advantation (Wilson B. S.
Data .	Contract of the provide states in the second states in the second states and second states in the second states in			T T T T T A F
unt-form (+				7
b furter				* Gr William
Autor	Parameter A			8 Galanteres
Li teutter				1 Accessive Selfs 45
# 8.000	Parameter 2			1 9.00 mm
g/ Unidation	and the second sec	The second s		T victore veters
al tracturan	lent d			- Strature
A montant				WebFarri Laupt
12 Deployed				
1.8 Latter				
GT Genored	Add			
Dean				
Dinner .				fragmental 11
P dedu				Resultant Sciences 11
T2 Ownerst				the latest
11 Relationstat				5. T. H.
d national				Carlas -
of large				Endland has
- Proved				lifest.
- Autobilia				Provider -
To Cample				Parght.
TARACE				Petergh I
T tells				Pastorio Pate
C. Beaund'station				Tellaho a
Ph Cargoniates				Test al
Companies -				let
104			-1	Tertori vise.
Cetceding	The second se			
in a la	A ready of the second of the second s			
Bents/Secel				

Figure 9.16: Design of the web form.

5. You need to add a reference to the VSService in order to use it in this application. To do this, select Project/Add Web Reference from the main menu or right-click on the project name (VSClient) in the Solution window and select Add Web Reference. On the left side of the window click on the hyperlink Web Reference on Local Web Server. This should show us all the discovery files (*.disco or *.vsdisco) from which we can select VSService.vsdisco. You might get a Directory Listing Denied message. In this case, type the full IIS path of the

VSService.vsdisco file and press Enter. You will see the Add Web Reference window as shown in Figure 9.17.

⊢ ¢ Ø 🖸 åddressi		n 🖉 🕫
The Add Web Reference benears helps you to backs online Web Services available from the only of Services available from the only of Services available from the only of Services available from the service devices on the Service available from the Service available from the Service available from the Service devices on the Service available from the Service devices on the Service device devices on the Service devices on the Service devices on the Service devices on the Service device devices on the Service device device device devices on the Service devices on the Service devices on the Service devices on the Service device device device device devices on the Service device device device devices on the Service device devic	Avalable seferences:	2
Web, Beferences, ch. Local Web, Server	-	<u>ح</u>

Figure 9.17: Adding a Web reference.

Visual Studio.NET will discover the web service and display results as shown in Figure 9.18.





Now click on the VSService.vsdisco hyperlink and then click on the Add Reference button at the bottom right of the window. A reference to the web service is added in the Solution Explorer as shown in Figure 9.19.

Solut	ion E	xplorer - ¥SClient	
		B	
		on 'VSClient' (1 project) SElient References Web References Service1.sdl Service.disco Config.web Global.asax Styles.css VSClient.disco	
	[1]	WebForm1.aspx	

Figure 9.19: Reference to web service in Solution Explorer.

- 6. Add the following code behind the click event of the button:
 - 7. Private Sub Button1_Click(ByVal sender As System.Object, _
 - 8. ByVal e As System.EventArgs) Handles Button1.Click
 - 9. Dim s As String
 - 10. Dim t As New localhost.Service1()
 - 11. s = t.add(CInt(Param1.Text), CInt(Param2.Text))
 - 12. ResultAdd.Text = s

End Sub

13. Now build and view the application in the browser by pressing F5. You will see the form as shown in Figure 9.20. Supply input values to the two textboxes Param1 and Param2 and click on the button. The click event of this button will make a call to the add method of the web service which returns the result of the addition. This gets displayed in the ResultAdd textbox.



Figure 9.20: Testing the Add function.

Using WebService Behavior to Make Function Calls

I want to show you how to make function calls using another methodology provided by Microsoft. This is the WebService Behavior. Remember that SOAP is a protocol and

ASP.NET is just a tool to implement it. SOAP is basically all about sending and receiving XML packets over HTTP. I do not need to depend on a tool to start using SOAP. However, having access to a prefabricated function library helps, because then I don't have to reinvent the wheel. The ASP.NET implementation of SOAP requires the existence of a proxy, whereas the WebServices Behavior does not. Both are sound technologies and my intent in presenting the WebServices Behavior is to enhance your SOAP toolkit, and in the process, present another way of using SOAP. The WebService Behavior is implemented with a lightweight HTML Components (HTC) file as an attached behavior that can be used with Internet Explorer 5.0 and above. The file WebServices.HTC can be downloaded from Microsoft's Web site at http://msdn.microsoft.com/workshop/author/webservice services can then be called using the Web page using the behavior. The web services can then be called using client-side JavaScript.

I have created a sample web service (behavior.asmx) to test out this technique. This file, together with the HTC file WebService.htc and two HTML files (populate.html and add.html) can be found in the behavior subfolder of the samples folder for this chapter. The web service contains two simple functions. The first is the add function, which we met in the preceding sections. The second function, populate, is a bit more interesting. As I had mentioned earlier, a web service using the SOAP protocol could be used to serialize an ASP.NET DataSet and I demonstrate this here. The Populate function takes as input parameters a connection string and a SQL Select query string. It returns back a DataSet containing the rows returned from the database after running the Select query. Here is the code extract for this function:

<WebMethod()> Public Function Populate(ConnStr as string, SQL as string) As DataSet

Dim dv As DataView

Dim i As integer

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

myConnection = New OleDbConnection(ConnStr)

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "vTable")

Populate = ds

End Function

The complete listing for the web service is as follows:

Behavior.asmx

<%@ WebService Language="VB" Class="Behavior"%>

Imports System

Imports System.Web.Services

Imports System.Data

Imports System.Data.OleDb

Imports System.Text

Public Class Behavior: Inherits WebService

<WebMethod()> Public Function TestFunction (vInput as Boolean) As String

If (vInput = TRUE) Then

TestFunction = "It is the truth..."

Else

TestFunction = "False!False!False"

End if

End Function

<WebMethod()> Public Function add(a as integer, b as integer) as string

add = cstr(a+b)

End Function

<WebMethod()> Public Function Populate(ConnStr as string, SQL as string) As DataSet

Dim dv As DataView

Dim i As integer

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

myConnection = New OleDbConnection(ConnStr)

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "vTable")

Populate = ds

End Function

End Class

The HTML form that makes a call to the <code>Populate</code> method of the web service is called Populate.html. Within this form, I make a call to the <code>Populate</code> function by passing to it a connection string and the SQL query (Select * from <code>Groups</code>) as input parameters. The method returns all the records from the <code>groups</code> table in XML format. Although I do not show it here, you can use an XSL stylesheet to render the returned XML. Figure 9.21 shows the response received from the <code>Populate</code> method call:



Figure 9.21: Calling the Populate function using Behavior.

The code listing of Populate.html is as follows: **Populate.html**

<html>

<head>

<script>

```
var idCall = null;
```

function init() {

WebServices.useService("behavior.asmx?WSDL","myService");

var vcn= "Provider=SQLOLEDB; Data Source=(local);";

```
vcn= vcn + "Initial Catalog=ASPNET;User ID=sa;";
```

```
var vSQL = "select * from Groups" ;
```

```
idCall = WebServices.myService.callService("Populate",vcn, vSQL);
```

}

```
function WebServices_OnResult() {
```

```
if (idCall == event.result.id)
```

if (!event.result.error)

txtResult.value = event.result.value.xml ;

else

txtResult.value = event.result.errorDetail.string;

}

</script>

</head>

```
<body onload="init()">
```

```
<div id="WebServices"
```

style="behavior:url(webservice.htc)"

```
onresult="WebServices_OnResult()">
```

</div>

Result:

<textarea id="txtResult" style="width:100%" rows="10"></textarea>

</body>

</html>

The WebService Behavior is attached to an element using the Style attribute. It is given an ID of <u>WebServices</u> so that it can be referenced from script.

<div id="WebServices"

style="behavior:url(webservice.htc)"

```
onresult="WebServices_OnResult()">
```

</div⊳

The Use method of the WebService maps the web service URL to a friendly name $m_{yservice}$. This alias can then be used to reference the web service in code, as in the following:

WebServices.useService("SQLService.asmx?SDL","myService").

The Populate function of SQLService.asmx expects two parameters: the connection string and the SQL string. These are stored in the variables vcn and vSQL respectively and passed on to the function, as in the following:

idCall = WebServices.myService.callService("Populate",vcn, vSQL)

The callService method initiates an asynchronous communication between the WebService Behavior and the web service. The OnResult event fires when the result of the call is received back as XML data packets. The following code then obtains the result and displays it in the text area of the screen.

Function WebServices_OnResult() {

If (idCall == event.result.id)

If (!event.result.error)

txtResult.value = event.result.value.xml ;

Else

txtResult.value = event.result.errorDetail.string;

}

The add() method of the web service accepts two integers as input parameters and returns an addition of the passed values as shown below:

Public Function <WebMethod()> add(a as integer, b as integer) as string

add = cstr(a+b)

End Function

The file $\underline{AddFunction.html}$ shows you how a call can be made to the $\underline{add}()$ function of the web service. The following is the listing of the file:

AddFunction.html

<html>

<head>

<script language="JavaScript">

var idCall = null;

function init() {

WebServices.useService("behavior.asmx?WSDL","myService");

var vcn= "Provider=SQLOLEDB; Data Source=(local);";

```
vcn= vcn + "Initial Catalog=ASPNET;User ID=sa;";
```

```
var vSQL = "select * from Groups" ;
```

idCall = WebServices.myService.callService("add",5,16);

}

```
function WebServices_OnResult() {
```

```
if (idCall == event.result.id)
```

```
if (!event.result.error)
```

txtResult.value = event.result.value ;

else

txtResult.value = event.result.errorDetail.string;

}

</script>

</head>

```
<body onload="init()">
```

<div id="WebServices"

style="behavior:url(webservice.htc)"

```
onresult="WebServices_OnResult()">
```

</div>

Result:


```
<textarea id="txtResult" style="width:100%" rows="10"></textarea>
```

</body>

</html>

This function call is similar to the Populate function call except that the event result is a value instead of XML.

Enabling Access Data Sources Across Domains

A common problem posted on XML/SOAP user groups is getting an access denied error from Internet Explorer while trying to call a web service across domains. This is because when using SOAP across domains, the Access data sources across domains option of Internet Explorer should be either Enabled or Prompt. To do this, choose Tools, Internet Options. Then select the Security tab. For the Internet settings, ensure that the option Access data sources across domains is either Enabled or Prompt, as per the screen shown in Figure 9.22.



Figure 9.22: Enabling the Access data sources across domains option of IE.

Summary

This chapter presented one of the most important topics in ASP.NET—web services. Although application-to-application communication techniques, such as DCOM, RPC, and MSMQ exist, they have not been very successful, because they required one to buy into a particular technology. Web services, on the other hand, are based on SOAP and XML, both of which can be understood by all machines and operating systems. In this chapter, I showed you how to use web services using a simple text editor as well as Visual Studio 7. I also showed you how to call web services using web service Behaviors. In <u>Project 2</u> (Chapters 18 through 22), I will show you how to build a database web service that will have generic functions for adding, updating, deleting, and selecting records from the database.

Chapter 10: ASP.NET Applications

An ASP.NET application corresponds to a virtual directory. All ASP.NET objects included in the same virtual directory comprise an ASP.NET application. These objects can be pages, web services, configuration files, global application files, assemblies, and application services such as security.

Creating a Virtual Directory

To create a new virtual directory in IIS under Windows NT, start Internet Service Manager. Right-click on an existing directory and choose New and then Virtual Directory (shown in Figure 10.1).



Figure 10.1: New virtual directory in IIS.

Promoting an Existing Folder to Be a Virtual Directory

You can promote an existing directory to be a virtual directory as follows:

- Right-click on the folder that you want to make a virtual directory.
 Under Application Settings, click the Create button next to the disabled
- Under Application Settings, click the Create button next to the disabled Default Application textbox (shown in <u>Figure 10.2</u>).



Figure 10.2: Create button.

3. The Create button will now be named Remove and the Name field will be enabled. The virtual directory is now ready (shown in <u>Figure 10.3</u>).

Finisher Finisher Grade Grade	Alt Programments Detectory Documents Dete	
	Loca Pat: ASPTest Bows Auces Permission Contex Contol Current Contol Dial Discovery Provide the Sectory Payload Districts Nape Discovery bowing allowed Payload Districts Nape Stating Pierc. Obtaint Veb Siles ASPT int Permissions Permissions Permissions Childrandia control of the Sectory District Control of the Sectory Configuration Permissions Childrandia control of the Sectory District Control of the Sectory Configuration Permissions Childrandia control of the Sectory District Control of the Sectory Configuration	
 avig avig2dent avig2dent	0K Cancel goods Help	

Creating a New Virtual Directory in Personal Web Server under Windows 2000

To create a virtual directory in Personal Web Server under Windows 2000, do the following:

Han A British	
3.0d/2kp	
B BARNER	effet
Text 3 Contang here	NT
Altarettede	
dercel	
P Engle Default Document	
Default Document(s): O efault htm:Default asp.instart.orp.Default aspx	_
C Altra Danton Research	_
F Save Web Site Activity Log	
-84.3% PB	

1. Start Personal Web Server and click on the Advanced button (shown in Figure 10.4).

Figure 10.4: New virtual directory in Personal Web Server.

2. Click on Add, browse to the folder, and give it a name under the Alias box.

The Global.asax File

The Global.asax file is quite similar in concept to the Global.asa file in ASP. Like Global.asa, the Global.asax file handles "application-level" logic through application events such as Application_Start, Application_End, Session_Start, and Session_End, to name a few. This file is dynamically parsed and compiled into a .NET Framework class the first time any resource within its application namespace is requested. This file cannot be requested directly by users, hence its contents are protected. The Global.asa and Global.asax files can coexist in the same virtual directory. This is to provide backward compatibility to ASP developers—ASP applications continue to use the Global.asa, whereas ASP.NET applications use Global.asax. However, the two files cannot share application and session state variables.

When you change the Global.asax file, the ASP.NET framework detects the change, completes all pending requests, and fires the Application_OnEvent event and reboots the application, which in turn clears all state information. This process is invisible to the user, who does not experience any downtime.

The Global.asax file has two events that are fired each time a request (or a postback) is made. These are the Application_BeginRequest and the

Application_EndRequest events. Code that must be executed at the beginning of each page can be placed in the Application_BeginRequest event. Figure 10.5 shows an example application that uses the Global.asax file. You can find the code in the "samples" folder for this chapter under the subfolder called "events." Please note that you must place the Global.asax file in the root directory of the virtual directory.

Global.asax

<script language="VB" runat="server">

Sub Application_Start(Sender As Object, E As EventArgs)

' Do application startup code here

End Sub

Sub Application_End(Sender As Object, E As EventArgs)

' Clean up application resources here

End Sub

Sub Session_Start(Sender As Object, E As EventArgs)

Response.Write("Session Start Fired...
")

End Sub

Sub Session_End(Sender As Object, E As EventArgs)

' Clean up session resources here

End Sub

Sub Application_BeginRequest(Sender As Object, E As EventArgs)

Response.Write("<h3> Global.asax Demo</h3>")

Response.Write("Begin Request Fired...
")

End Sub

Sub Application_EndRequest(Sender As Object, E As EventArgs)

Response.Write("End Request Fired...
")

End Sub

</script>

Here is a Web page that calls it. GlobalTest.aspx

<html>

```
<script language="VB" runat="server">
```

Sub Page_Load(Sender As Object, E As EventArgs)

Response.Write("Page.Load Fired...
")

End Sub

Sub Session_End(Sender As Object, E As EventArgs)

Session.Abandon()

Response.Redirect("GlobalTest.aspx")

End Sub

</script>

<body>

<form runat="server">

<input type="submit" Value="Refresh " runat="server"/>

<input type="submit" OnServerClick="Session_End" Value="End Session" runat="server"/>

<hr>

</form>

</body>

</html>

When the page is requested, the following sequence of events occurs:

- 1. Application_BeginRequest fires.
- 2. Session_Start fires.
- 3. Page Load fires.
- 4. Application_EndRequest fires.

Fic	ure	10.5	shows	this
1 10	iui c	10.0	0110103	

Andrew Contract	10-120010										-	30.
112 (-Q -	0	Petrah	Gene	G. Seatch	(avoite:	C.	Na.	- Ber	The second secon	Mecange	C. N
Global.asa	x Demo											
legin Reques Territon Start Page Lond Fr	t Fired Fired											
Ratech	End Sec	cios										_
Ratech	End See	cios										-
Ratech	End Sec Feed	cios										-
Ratech	End Sec	cion										-
Rotech	End Sec	tion										-
Ratech	End Sec	tios										-
Ratuch	End Sec	tion										

Figure 10.5: Initial request.

When the page is refreshed, the following sequence of events occurs:

- 1. Application_BeginRequest fires.
- 2. Page Load fires.
- 3. Application_EndRequest fires.

Figure 10.6 shows this.

giden (P)	Mip.//127.001	NGPVirtual	'ghbeTexte	ipk -							-	20
Ģ. Back	- op	0	Petroh	Hone	G. Snarch	favoite:	C. Haloy	NN.	(B) Pix	1 I I I I I I I I I I I I I I I I I I I	Meconger	
lobal.s	sax Demo	6										
egin Req age Load	wat Fired											
Retesh	EndiSe	ssion										
dlama	at Frend											
20 24 90	e: 2290											
									and the second se			

Figure 10.6: Page refresh.

When the session is abandoned, the following sequence of events occurs:

- 1. Application_BeginRequest fires.
- 2. Session_Start fires.
- 3. Page Load fires.
- 4. Application_EndRequest fires.

When the session is abandoned, a new session is created and the Session_Start event is fired again. Figure 10.7 shows this.

Mp.//12	7.0.0 LASPY	intend/Clob	ell'est esp	s - Microsof	A laternet	Explorer					-	5
Elo Lot	You Fgra	ant Toop	Fields									107
Agdess (4)	Mp//127001	Call Contract	Gbballeste	A	R	(in	a	1.2.	Ab	1	-	C- 00
Back.	- conora	5/00	Fetzh	Hone	Search	Favoites	Holay	Mai	Pier	10	Meconger	1
Global.	asax Demo											1
Begin Re	ouest Fired											
Seraina S	hat Fred											
age Loa	d Fred											
Datash	Endler	antine										
	010 01											- 1
Sa4 Requ	est Fred.											
] Done										1011	ernet	
AStat 3	AP. WH C	SE QE	18410	121 00	10010	A 21	(Maria)	CIER	34 - 18	0.40	35 F 25	IS PH

Figure 10.7: Session abandoned.

Locking and UnLocking methods provide a safeguard against data corruption when multiple users are trying to update the same variable. The syntax for this is straightforward as shown in the following example:

<%

```
Application.Lock()
Application("hits") = CType(Application("hits") + 1, Int32)
Application.UnLock()
```

%>

When a page finishes execution or times out or an unhandled error occurs, the Application object is automatically unlocked.

Storing data in Application objects should be used wisely because significant resources, which might be put to better use elsewhere, might be absorbed. The Application object is not maintained across Web farms or Web gardens, which might be a limitation if you use these techniques.

Application or Session-Scoped Objects

You can define .NET Framework classes or classic COM components with either an appinstance, session, or application scope using the object tag. The appinstance scope implies that the object is specific to one instance of HttpApplication and is not shared.

<object id="id" runat="server" class=".NET Framework class Name" scope="appinstance"/>

<object id="id" runat="server" progid="Classic COM ProgID" scope="session"/>

<object id="id" runat="server" classid="Classic COM ClassID" scope="application"/>

Global.asax and Application State

You can store variables with application-level scope in the Global.asax file. The following example shows you how to store a DataView in an application variable, which can then be used throughout an application. The code for this example is available in the "..samples\Application" subfolder on the book's Web site at

www.premierpressbooks.com/downloads.asp.

First code the Application_Start event of the Global.asax file as follows:

Global.asax

<%@ Import Namespace="System.IO" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<script language="VB" runat="server">

Sub Application_Start(Sender As Object, E As EventArgs)

' Do application startup code here

Dim dv As DataView

Dim i As integer

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial Catalog=ASP NET;User ID=sa;"

```
myConnection = New OleDbConnection(ConnStr)
```

SQL = "select * from groups "

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "groups")

dv = new DataView(ds.Tables("groups"))

Application("Source") = dv

End Sub

</script>

I have extracted rows from the Groups table and populated a DataView with the resultset. I have then stored this DataView in an application variable called source. Note that I had to import System.Data and the System.ADO namespaces as I have to interact with the database.

I can now use the source application variable to populate a DataGrid in a web form as follows:

ApplicationState.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server">

Sub Page_Load(Src As Object, E As EventArgs)

Dim Source As DataView = Application("Source")

vSpan.InnerHtml = Source.Table.TableName

vGrid.DataSource = Source

vGrid.DataBind()

End Sub

</script>

<body>

<h3>DataSource in Application_OnStart</h3>

<h4> Table: </h4>

<ASP:DataGrid id="vGrid" runat="server"

Width="900"

BackColor="#ccccff"

BorderColor="black"

ShowFooter="false"

CellPadding=3

CellSpacing="0"

Font-Name="Verdana"

Font-Size="8pt"

HeaderStyle-BackColor="#aaaadd"

MaintainState="false"

/>

</body>

</html>

(he galeo	Life year (parties just 1946) Comparty 11 and Princel application of the sep Day Prince Research States	G D G C	- Brit House	
ab	aSource in Application_OnSt le: groups	art		
1	ande, destes	unde_value	unde, category	1-per
	CRETE MODUNE	1		L
	Long term leans	1		1.
	Correct Rabilities	3		
4	First arrats	4		4
	Investments	1		A
	Current essets	8		s
	Revenue accounts	2		1
	Branch/divisions		1	*
	Reserves and surplus	8		1. 1. 1. 1.
	Secured loave	2018	1	4
	Shoebared loans	299	4	5
	Cutes takes payable (In)	340	5	4
	Printages	308	3	L
	Surdry and/ors	302	2	L
	derk of 6 limits	303	2	L
1	Exposite (assets)	808	4	4
	Advances (access)	642	4	4
	Surdre debture	645	4	4
	Centric hand	104	4	4
	Baru scouts	ki05	4	4.
	Lake account	700	P	1 0
	Purchase account	248	P.:	1
	Income account	742	1.	() () () () () () () () () ()

Figure 10.8: Application state.

Session State

While the Application object has not changed much from the previous versions of ASP, the Session object has seen a major updating. A session variable is a key-value pair that can be set and read for the duration of a user's session. For example, you can store a value in a session variable as follows:

Session("Name") = "Hersh Bhasin"

This session variable can then be accessed on a Web page as follows:

Dim ls_name As string

ls_name = Session("Name")

Each session is assigned a unique key, which is stored as an HTTP cookie and sent to the server on each request. The server reads the key and reestablishes the server state. The problem with this implementation is that clients might reject cookies for privacy or security reasons, which in turn disables state management. Another problem is that the state management is linked with the Web server (IIS) and when the Web server is recycled or fails, state information is lost. The third disadvantage is that each ASP server maintains its own state and unless the user returns to the same server, state will be lost. This would be the case where ISPs use proxy load balancing solutions.

ASP.NET solves these problems. The ASP.NET session state is process independent. This means that even if the Web server goes down or is restarted, state will still be maintained. The session content can alternatively be serialized to a SQL Server database and the state information can be reloaded after a machine failure. Using session variables has always been a problem on Web farms where multiple servers handle user requests. This is not a problem anymore. By moving to an out-of-process model, ASP.NET enables servers in the farm to share a session state process.

Session state settings are handled by settings in the web.config file. This file will be discussed in greater detail later in this chapter. The default file (called machine.config) is located in the ...\Framework\[Version]\ folder of the windows folder (for example the WinNt\Framework\[Version]\ folder). Each application can have its own web.config file and the settings specified there override the default machine.config file.

Six settings can be applied to configure session state (<sessionState>). These are

- mode: This can be InProc, SQLServer, and StateServer. The InProc stands for in-process and is the traditional ASP state management setting. There are two types of out-of-process modes: memory-based (StateServer) and SQL Server-based (SQLServer). I will be dealing with these in detail later.
- **cookieless:** This can be either True or False. The default is false. True implies that this mode is enabled.
- timeout: Length of time that a session is considered valid.
- sqlConnectionString: Used with the SQL Server mode. Specifies the connection string to the ASP.NET state management database.
- StateConnectionString: Used when the mode is StateServer. This identifies the TCP/IP address and port of the Windows NT Service that provides the state management facilities.

I have created a test web form that I will be using in my discussion. This form has two simple functions, one for setting a session state variable and the other for retrieving the value of this variable. Here is what the code looks like:

```
Session.aspx
```

<html>

<Script runat="server">

Sub AddSession(sender As Object, e As EventArgs)

Session("MySession") = text1.Value

message.text = "Session Variable stored as : " +Session("MySession").ToString()

End Sub

```
Sub CheckVariable(sender As Object, e As EventArgs)
```

```
If Session("MySession") = "" Then
```

message.text = "Session Data has been erased"

Else

```
message.text = "Stored Session Variable is : " + Session("MySession").ToString()
```

End If

End Sub

</Script>

<body style="font: 10pt verdana; background-color:ivory">

<h1> Session State </h1>

<form runat=server>

<input id=text1 type=text runat=server>

<asp:Button id="Add" text="Add Session Variable" onclick="addSession" runat="server" />

<asp:Button id="Check" text="Check Session Variable" onclick="checkVariable" runat="server" />

<hr>

```
<asp:label id="message" runat="server" />
```

</form>

</body>

</html>

In-Process Mode

The in-process mode is the way ASP has always handled state management. State is managed within the process and is lost when the process is recycled. You might question the need for this mode when you have the out-of-process modes available, which are undoubtedly more efficient. The reason is performance. Out-of-process modes add additional overhead required to marshal data back and forth between processes or SQL Server. The following are the steps you will take to set up state management in this mode:

- 1. In web.config, set the mode attribute of sessionState to InProc (this is the default). When this setting is chosen, the only other web.config settings used are timeout and cookieless.
 - 2. <configuration>
 - 3. <system.web>
 - 4. <sessionState
 - 5. mode="InProc"
 - 6. cookieless="false"
 - 7. timeout="20"
 - 8. />

9. </system.web>

</configuration>

10. Run Session.aspx. Store a session state value. Stop and restart the IIS using the command-line utility iisreset, which ships with IIS5 as follows:

C:\iisreset [computername] /RESTART (or simply issue the command iisreset from the command prompt)

11. Click the Check Session Variable button. You will have lost the session variable.

The code for this example is included in the ... $\$ samples $\$ sessionstate in process subfolder for this chapter on the book's Web site at

www.premierpressbooks.com/downloads.asp.

Out-of-Process Mode

Out-of-process mode maintains state in a separate process. This mode gives the reliability of a separate process with the performance advantage of reading and writing from memory. This state should be used when performance is important but when you can't guarantee which server a user uses to request an application. Here are the steps to set up this mode:

- 1. In web.config, set the mode attribute of sessionState to
 - "StateServer." This tells ASP.NET to look for the ASP.NET state service at the server and port settings specified in the web.config file, which in this case is the local server.
 - 2. <configuration>
 - 3. <system.web>
 - 4. <sessionState
 - 5. mode="StateServer"
 - 6. stateConnectionString ="tcpip=127.0.0.1:42424"
 - 7. sqlConnectionString="data source=127.0.0.1;user
 - id=sa;password="
 - 8. cookieless="false"
 - 9. timeout="20"
 - 10. />
 - 11. </system.web>

</configuration>

- 12. The ASP.NET SDK includes Windows NT service called
 - aspnet_state, which is used by ASP.NET for out-of-process state management. Start this service by running this at the command prompt:

net start aspnet_state

Figure 10.9 shows what you will see.



Figure 10.9: Starting ASPState service.

- 13. Run Session.aspx. Store a session state value. Stop and restart the IIS using the command-line utility iisreset.
- 14. Click the Check Session Variable button. You will note that you have retained the Session variable.

The code for this example is included in the ...\samples\sessionstate\OutofProcess subfolder for this chapter on the book's Web site at www.premierpressbooks.com/downloads.asp.

SQL Server Mode

This mode should be used when reliability considerations are paramount. The database can be clustered for handling server failures. The trade-off is performance as this mode is slower than out-of-process. You can set up this mode as follows:

- 1. Create the ASPState Database by applying the file InstallSqlState.sql provided with ASP.NET and located at
 -\Microsoft.NET\Framework\[version]\ folder with the utility osql.exe as follows:
- osql –S [server name] –U [user] –P [password] < InstallSqlState.sql

You can also directly open this file in the SQL Query Analyzer tool (ISQL) of Microsoft SQL Server and execute it. (I find this method better because I don't have to remember passwords, server names, and so on. I can just log on with system administrator rights and execute the script.)

This process will create a database called ASPState, which contains 16 stored procedures and two tables in tempdb. These are

AspStateTempSessions and AspStateTempApplications. Now you must stop and restart SQL Server as some startup procedures were created which need to run.

A script to uninstall the ASPState database is also provided. This script is called UninstallSqlState.sql.

- 2. In the configuration file, set the mode attribute of sessionState to "SQLServer".
 - 3. <configuration>
 - 4. <system.web>
 - 5. <sessionState

- 6. mode="SQLServer"
- 7. stateConnectionString ="tcpip=127.0.0.1:42424"
- 8. sqlConnectionString="data source=127.0.0.1;user id=sa;password="
- 9. cookieless="false"
- 10. timeout="20"
- 11. />
- 12. </system.web>

</configuration>

- 13. Run Session.aspx. Store a session state value. Stop and restart the IIS using the command-line utility iisreset.
- 14. Click the Check Session Variable button. You will note that you have retained the Session variable.
- 15. Connect to the tempdb database and run the following queries:
 - 16. select * from AspStateTempSessions

select * from AspStateTempApplications

You will see that the data stored in the SQL server tables AspStateTempSessions and AspStateTempApplications. Note that a unique session id is stored in the database. Figure 10.10 is a sample of the output.

relect * from Asp2tateTer	M + .	V 3 =	() tempifi		ICA 28		
SessionId	Created	Ispires	Lookbate	LoskCookie	Timecut	Looked	SecolonItem
hext0d555411bt45fag	2001-03-0	2001	2001	2	20	0	0x140000000
mwyma5ylwbat5tdi	2001-00-0	2001	2001	2	20	0	0x140000000
Ape Id Appliant							
1 /LR/WHVC/1/Poot/	ANDVirtual						
tere fill stereout							

Figure 10.10: State maintained in SQL Server.

We could further enhance the reliability of storing session state by clustering SQL Servers so that if one server becomes unavailable, another one replicating it takes over.

Cookieless State

This feature of the ASP.NET session state allows clients who turn off cookies to take advantage of the session state. In this mode, the session id is "munged" into the URL as follows:

http://localhost/(hmxz0d5554l1bt45faqudj55)/Application/Sessione.aspx All you have to do to enable this state is set the cookieless attribute to true as follows:

<configuration>

<system.web>

<sessionState

mode="StateServer"

stateConnectionString ="tcpip=127.0.0.1:42424"

```
sqlConnectionString="data source=127.0.0.1;user id=sa;password="
cookieless="false"
timeout="20"
/>
</system.web>
</configuration>
```

All modes are supported for cookieless sessions.

The Configuration File

ASP.NET can be configured using two configuration files. These are the machine.config and the web.config files. Each machine has a single machine.config file that holds the default configuration information. Each application can have its own optional web.config file and when such a file exists, it overrides the default machine.config file located in the WinNt\Framework\[Version]\ folder. The "local" web.config file can have only the sections it needs. These sections then override the default machine.config file. The sections not specified in the local web.config file are read from the machine.config file. These configuration files are an XML-based text file. A sample configuration file with two settings is shown here:

<configuration>

<system.web>

<sessionState

mode="StateServer"

stateConnectionString ="tcpip=127.0.0.1:42424"

sqlConnectionString="data source=127.0.0.1;user id=sa;password="

- cookieless="false"
- timeout="20"

/>

<appSettings>

<add key="dsn" value ="Bhasin;uid=sa;pwd=""/>

</appSettings>

</system.web>

</configuration>

This file cannot be accessed directly from a browser. The above example shows a file with two configuration sections <<u>sessionState</u>> and <appSettings>. I have already discussed the sessionState section. I will now discuss some other sections contained in this file.

<Configuration>

This is the root element and all configuration sections must be contained within the Configuration tags.

<appSettings>

The appSettings section allows you to use the web.config file as an ini file. You can store application-wide settings here. For example, you can store the DSN name here, which then becomes available to all the pages in the application. The way to do so follows.

In web.config, set this setting as follows:

<appSettings>

<add key="dsn" value ="Bhasin;uid=sa;pwd=""/>

</appSettings>

This section has one sub-element, which is add. The add sub-element supports two attributes: key and value. The key is the *name* of the variable you store in the value attribute. You can have as many add sub-elements as you require.

On a Web page you will extract the stored value, using Visual Basic.NET as follows: AppSettingVb.aspx

<html>

<script language="VB" runat="server">

Sub Page_Load(Src As Object, E As EventArgs)

Dim dsn As String = ConfigurationSettings.AppSettings("dsn")

response.write(dsn)

End Sub

</script>

</html>

You can find this code in the ...\samples\configFile\configAppSettings folder on the book's Web site at <u>www.premierpressbooks.com/downloads.asp</u>.

Here is the code in C#: **AppSettingC.aspx**

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

public void Page_Load(Object sender, EventArgs e)

{

String s = ConfigurationSettings.AppSettings["dsn"];

dsn.InnerHtml =s;

}

</script>

The DSN entry in web.config is:

<Compilation>

This section has one attribute (debug) and two subsections (<compilers> and <assemblies>). The debug attribute is a boolean setting that can be True or False. Setting it to True will compile in debug mode, which is slower and should be turned off in production sites.

In the compiler section, you set the language and extension attributes for each language you will use. Say you associate extension "vb" with the language Visual Basic. All web forms with this extension will have Visual Basic as the default scripting language and you now do not have to declare this language in page directives or script tags. The <assemblies> subsection enables you to specify which assemblies you want to include when compiling a resource. If you remember from previous chapters, when creating business objects (Chapter 8) and custom controls (Chapter 7), we had to include various assemblies in the bat files which compiled the objects we created. Instead of doing that, we can specify these assemblies here. Remember, however, that we still need to import these assemblies in Web pages using page directives. The <assemblies> sub-element has three more sub-elements: add, remove, and clear. The add sub-element adds assemblies and you can use the * to add all assemblies located in the /bin directory. Remove removes an assembly reference and the clear sub-element removes all references contained in or inherited from the main config.web.

<compilation debug="false" explicit="true" defaultLanguage="vb">

<compilers>

```
<compiler language="c#;cs;csharp" extension=".cs"
type="Microsoft.CSharp.CSharpCodeProvider,
```

System, Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"

/>

</compilers>

<assemblies>

<add assembly="mscorlib"/>

<add assembly="System, Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>

<add assembly="*"/>

</assemblies>

</compilation>

<Custom Errors>

ASP.NET provides a lot of helpful information when an error occurs, and while this may be helpful to the developers, it might scare the users of the site. The Custom Error setting allows you to specify a custom error page, which displays your custom error messages. You can also redirect specific errors (say 404 or 500 errors) to separate custom error pages.

The custom error section has two attributes. These are:

- defaultRedirect: Specify your custom error page here.
- mode: This can be On, Off, or RemoteOnly. The On mode redirects all errors to the custom error page. The Off mode turns off custom errors, and the RemoteOnly mode turns on the custom errors only for remote users—if you are on the same server and generate an error, you see the full details of the error. This option is helpful because the developers see the complete details of the error whereas users of the site see only the custom errors.

I will now show you how to create a custom error page. You can find the code in the..samples\configFile\CustomErrors subfolder on the book's Web site at www.premierpressbooks.com/downloads.asp.

web.config for Custom Errors

<configuration>

<system.web>

<customErrors mode="On" defaultRedirect="HandleError.aspx" >

</customErrors>

</system.web>

</configuration>

I have specified my custom error file to be HandleError.aspx, the listing of which is as follows:

HandleError.aspx

<%@Page Language="VB"%>

<html>

<head>

<style>

body {font-family:Tahoma,Arial,sans-serif; font-size:10pt}

</style>

</head>

<body>

<h3>Default Custom Error Page</h3>

There was an error in the page

Return to the previous page

</body>

</html>
I have created a test page and I try to navigate to a non-existent page in this file, thus generating an error. Here is the listing:

ErrorTest.aspx

<html>

<head>

<script language="VB" runat="server">

Sub cause_error(Source As Object, E As EventArgs)

Response.Redirect("NonExistantPage.aspx")

End Sub

</script>

<form runat="server">

<asp:Button id="Error" text="Cause Error" onclick="cause_error" runat="server" />

</form>

</head>

</html>

The <customError> section supports one subsection: the <error> sub-element. You can use sub-element to redirect specific error numbers to their own custom error pages. For example:

<configuration>

<system.web>

<customErrors mode="On" defaultRedirect="HandleError.aspx" >

<error statusCode = "500" redirect ="500Error.aspx"/>

</customErrors>

</system.web>

</configuration>

All errors with a status code of 500 will now be redirected to the custom error page 500Error.aspx.

<Globalization>

The Globalization subsection has five attributes. These are:

- requestEncoding: This is the default encoding for all requests.
- responseEncoding: This is the default encoding for all responses.
- fileenCoding: This is the default encoding for all .aspx, asmx, and aspc files.
- culture: This is the default culture used to process requests. This is information regarding language, calendar, and writing system. For

example, English = en. Information pertaining to cultures can be found under the namespace System.Globalization in the CultureInfo class.

 uiCulture: This is the default culture to lookup resources and also expects a CultureInfo value like en.

Here is an example:

<globalization

fileEncoding="iso-8859-1"

requestEncoding="iso-8859-1"

responseEncoding="iso-8859-1"

culture="en"

uiCulture="en"/>

<httpHandlers>

 $\tt httpHandlers$ allow you to map incoming requests to .NET Framework classes that can handle those requests. For example:

<httpHandlers>

```
<add verb="*" path="*.aspx" type="System.Web.UI.PageHandlerFactory" />
```

</httpHandlers>

This setting tells the .NET Framework that all requests for the file with an extension of aspx should be handled by the .NET Framework class System.Web.UI.PageHandlerFactory.

As you can see, this section has three attributes. These are:

- verb: This attribute tells the .NET runtime to process the request using a GET, POST, or PUT, or all three separated by a comma (an * implies the same thing).
- path: The path to a file (or a set of files with the same extension) that needs to be processed.
- type: The name of the assembly or class that will handle the request.

You can write your own httpHandlers. Suppose that you want to deny access to web forms called default.aspx. You would first write a simple class to handle requests to this form as follows:

handler.vb

Imports System.Web

Namespace Hersh

Public Class CustomHandlerVB : Implements IHttpHandler

Public Sub ProcessRequest(Context As HttpContext) Implements IHttpHandler.ProcessRequest

Context.Response.Write("Sorry! Access to this resource is Denied...")

End Sub

Public ReadOnly Property IsReusable As Boolean Implements IHttpHandler.IsReusable

Get

Return true

End Get

End Property

End Class

End Namespace

A custom HTTP Handler is created using the IHttpHandler interface. This interface contains only two methods: IsReusable and ProcessRequest. The IsReusable method tells the HTTP factory whether the same instance can be used to serve multiple requests, and ProcessRequest takes the HttpContext instance as a parameter which, in turn, makes the response and request intrinsics-accessible. In the preceding example, the request data is ignored and a string is sent back as a response.

You should now compile this class as follows:

make.bat

set outdir=g:\aspNetSamples\bin\CustomHandlerVB.dll

set assemblies=System.Web.dll

vbc /t:library /out:%outdir% /r:%assemblies% handler.vb

pause

Make sure that the outdir parameter points to your local /bin directory.

Add the following section to the web.config file:

<configuration>

<system.web>

<httpHandlers>

<add verb="*" path="default.aspx"

type="Hersh.CustomHandlerVB,CustomHandlerVB" />

</httpHandlers>

</system.web>

</configuration>

Finally test it out by trying to open a file called default.aspx through IIS. You will get a message that says "Sorry! Access to this resource is denied. . . . "

The complete source for this example is provided in the ...samples\configfile\httpHandler subfolder on the book's Web site at <u>www.premierpressbooks.com/downloads.asp</u>.

<httpModules>

This section enables you to configure HTTP modules for your application. Most of the classes and assemblies that you will need are included by default. You can use this section to add your own handlers.

<httpModules>

<add name="OutputCache" type="System.Web.Caching.OutputCacheModule, System.Web, Version=1.0.2411.0,

Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />

<add name="Session" type="System.Web.SessionState.SessionStateModule, System.Web,

Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />

```
<add name="WindowsAuthentication"
```

type="System.Web.Security.WindowsAuthenticationModule,

System.Web, Version=1.0.2411.0, Culture=neutral,

PublicKeyToken=b03f5f7f11d50a3a" />

</httpModules>

<processModel>

This subsection allows you to configure the process model of your Web applications. A number of settings can be configured. For example:

<processModel

enable="true" timeout="Infinite" idleTimeout="Infinite" shutdownTimeout="0:00:05" requestLimit="Infinite" requestQueueLimit="5000" restartQueueLimit="10" memoryLimit="60" webGarden="false" cpuMask="0xffffffff" userName="SYSTEM" password="AutoGenerate" logLevel="Errors" clientConnectedCheck="0:00:05" comAuthenticationLevel="Connect" comImpersonationLevel="Impersonate" /> The meaning of the attributes is as follows: enable: The Boolean which specifies whether this element is enabled.

- timeout: The number of minutes after which a new IIS worker process is launched to replace the current one.
- idleTimeout: The number of minutes the worker process can remain idle before timing out.
- shutdownTimeout: The number of minutes a worker process has to gracefully shut down before it is killed.
- requestLimit: The number of requests a worker process can handle before it is shut down and a new one is created.
- requestQueueLimit: The number of requests allowed to accumulate in the request pool before it is signaled that the worker process is bad and a new process is started to replace it.
- memoryLimit: The percentage of memory that can be used before the process is shut down and restarted.
- cpuMask: For Web gardens, it controls the number of processes.
- WebGarden: For Web gardens, it controls cpu affinity.

You might want to experiment with different settings of the memoryLimit to try to increase site performance. A server with low memory would reach the default setting of memoryLimit (40 percent) very rapidly, which would cause the process to be restarted and stopped in a loop.

Summary

In this chapter, you learned about the concept of applications in an ASP.NET setting. You also learned about the Global.asax file and its implications in maintaining variables with application scope. Maintaining variables in the Session State has undergone a radical change in ASP.NET and those issues were addressed. Finally you learned more about the web.config file and its major subsections. Two important subsections of the web.config file are expanded upon in separate chapters. The topics are tracing, which is discussed in <u>Chapter 12</u>, and security, which is discussed in <u>Chapter 13</u>.

Chapter 11: Caching

Overview

Caching is an important technique in building Web sites that are fast and efficient. The theory behind caching is that there are some items of the Web site that are very expensive to construct and that such items should be created once and then stashed away in memory for a fixed duration of time or until they change. Subsequent calls to these resources will not re-create the resource but simply retrieve it from the cache. Such items are typically resources that remain unchanged over a period of time—for example, shopping lists and price lists. The ASP.NET framework itself uses caching. When an initial request is made for an ASP page, it is compiled as an instance of the page class and cached on the server. Subsequent requests for this page loads the cached version until the page is modified or the caching period expires. At that time, the cache is updated with the changes.

ASP.NET supports the following two types of caching:

- Output Caching
- Data Caching

Output caching is the process of caching an entire page. This technique is useful for sites with heavy traffic where frequently accessed pages are stashed away in the cache. For caching to work, the page requested by different users must be identical in all respects. If requests are not for identical pages, the pages cannot be pulled out from the cache but must be regenerated for each request. For this reason, output caching works with GET requests (and query strings) but not with POST. With a GET request, the ASP runtime can look at the querystring, and all requests with identical querystrings receive a

cached version of the resource. Data caching involves identifying objects or data on a page that are expensive to construct and caching only those objects.

Output Caching

A simple page directive at the top of a web form accomplishes output caching as follows:

<%@ OutputCache Duration="10" %>

This directive means that the page will be remembered for 10 seconds. The first request for this page will store the page in the cache, and after that for ten seconds any number of users requesting this page will be served the cached page. This will, in turn, improve the site performance drastically. The following is an example:

MastersGrid.aspx

<%@ OutputCache Duration="60" VaryByParam="none" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<head>

<H4>Masters Table</H4>

<script language="VB" runat="server">

Sub Page_Load(Source As Object, E As EventArgs)

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

'Connection syntax

ConnStr = "Provider=SQLOLEDB; Data Source=(local);"

ConnStr = ConnStr + " Initial Catalog=ASPNET; User ID=sa"

myConnection = New OleDbConnection(ConnStr)

'DataSetCommand

SQL = "select * from Masters"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "Masters")

'Binding a Grid

DataGrid1.DataSource=ds.Tables("Masters").DefaultView

DataGrid1.DataBind()

'Display the creation time

TimeMsg.Text = DateTime.Now.ToString()

End Sub

</script>

</head>

<body style="background-color='beige'; font-family='verdana'; font-size='10pt'">

<form runat=server>

<asp:DataGrid id="DataGrid1" runat="server" />

<i>Last generated on:</i> <asp:label id="TimeMsg" runat="server"/>

</form>

</body>

</html>

This web form binds a DataGrid to a DataSet, which is populated initially from the database. All subsequent page requests for the next 10 seconds are met from the cache. The "last generated" time shows you the time the page was initially generated. Subsequent browser refreshes (for the next 10 seconds) show the same time, indicating that the page is being served from the cache. If a querystring matches in all respects, then the page is rendered from the cache; otherwise, it is re-created as the following example shows:

querystring.aspx

<%@ OutputCache Duration="60" VaryByParam="none" %>

<html>

<script language="VB" runat="server">

Sub Page_Load(Src As Object, E As EventArgs)

Dim querystring As String

querystring = Request.QueryString("name")

nameMsg.text = querystring

TimeMsg.Text = DateTime.Now.ToString()

End Sub

</script>

<body>

<h3>QueryString & the Output Cache</h3>

<table cellspacing="0" cellpadding="3" rules="all"

style="background-color:#AAAADD;

border-color:black;border-color:black;width:700px;border-collapse:collapse;">

Hersh

Bhasin

John

Smith

Bob

<i>name:</i> <asp:label id="nameMsg" runat="server"/>

<i>Last generated on:</i> <asp:label id="TimeMsg" runat="server"/>

</body>

</html>

In this example, the form posts back to itself and passes back a querystring, which contains a person's name. A time is displayed at the bottom, which shows when the page was created. If you select a name initially, it creates the timestamp at the bottom. If you now reselect it, the time remains unchanged, showing that the page was extracted from the cache. If you select a different name, the time changes again (because the querystring changes).

Page Data Caching

ASP.NET provides a robust caching engine that we can use to cache specific objects. Whereas output caching involved caching an entire page, data caching involves caching certain objects on the page, which are expensive to construct, and would thus benefit from being cached. The ASP.NET cache is private to each application, and its lifetime is the lifetime of the application. It is global to an ASP.NET application and is thread-safe as it implements automatic locking, thus concurrent users can access it. Its syntax is much like the application and session objects as the following code shows:

Adding values to the cache:

Cache("MyKey") = "SomeValue" • Or alternatively you can use the following:

Cache.Insert("MyKey","SomeValue") • Extracting values from the cache:

Dim Is_string as string

Ls_string = Cache("MyKey") • Removing values from the cache:

Cache.Remove("MyKey")

File and Key Dependencies

Web sites will often store information in XML files. Some of these files will not need to be changed for a period of time. For example, a product catalog, or price list, would only change when a new product is added or modified. It would be a waste of resources to generate such files dynamically each time that they are requested. Developers have often resorted to batch updates of such files (a fresh copy of the XML file is regenerated each time a change is made to the file and stored on the server). Requests are then directed to this static file. This process, however, requires manual intervention, with all its associated problems. ASP.NET caching techniques enable you to set up a link or "dependency" between the XML file and the file stored in the cache so that each time the XML file is changed, the cache will be refreshed, else all requests for this file will be met from the cache.

Consider the site navigation user control that was developed in <u>Chapter 6</u>, "<u>User</u> <u>Controls.</u>" I built a user control, which read an XML file that contained the site's navigation links and built the site navigation menu. Now the site navigation structure is fairly static; that is, I want to show the same navigation structure to all users, and it is a waste of resources to dynamically create it for each user request. Thus, I can safely cache the site navigation data and only refresh the cache if I change the site navigation XML file. I will now modify the control described in <u>Chapter 6</u> to take advant age of caching. The source code for this example can be found in the dependencies subfolder on the book's Web site at <u>www.premierpressbooks.com/downloads.asp</u>. The following is the XML file that holds my site navigation links:

nav.xml

<Siteinfo>

<site>

<sitename>Home</sitename>

<siteurl>default.aspx</siteurl>

</site>

<site>

<sitename>Masters</sitename>

<siteurl>masters.aspx</siteurl>

</site>

<site>

<sitename>Transactions</sitename>

<siteurl>Transactions.aspx</siteurl>

</site>

<site>

<sitename>Trial Balance</sitename>

<siteurl>trialbalance.aspx</siteurl>

</site>

</Siteinfo>

The user control is modified so that a dependency is set up between the cache and the XML file, as follows: nav.ascx

<% @ Import Namespace="System.Data" %> <% @ Import Namespace="System.IO" %> <% @ Import Namespace="System.Drawing" %> <script language="VB" runat="server"> 'Public Variable for each exposed Property PUBLIC vGridLines As GridLines PUBLIC vGorderColor as String PUBLIC vCellPadding As Integer Sub Page_Load(Src As Object, E As EventArgs) If Not IsPostBack LoadData() End If End Sub Sub LoadData Dim ds As New DataSet Dim fs As filestream

Dim xmLStream As StreamReader

Dim Source as DataView

Source = Cache("MyData")

If Source is Nothing

```
fs = New filestream(Server.MapPath("nav.xml"), FileMode.Open, FileAccess.Read)
```

```
xmlStream = new StreamReader(fs)
```

ds.ReadXML(XmlStream)

fs.Close()

Source = New DataView(ds.Tables(0))

'cache it for future use

Cache.Insert("MyData", Source, New CacheDependency(Server.MapPath("nav.xml")))

' we created the data explicitly, so advertise that fact

CacheMsg.Text = "Dataset created explicitly"

Else

CacheMsg.Text = "Dataset retrieved from cache"

End If

dlist.DataSource=Source

dlist.DataBind()

dlist.GridLines = vGridLines

dlist.BorderColor=System.Drawing.Color.FromName(vBorderColor)

dlist.CellPadding=vCellPadding

End Sub

</script>

<asp:DataList runat=server id="dlist"

RepeatDirection="horizontal"

RepeatMode="Table"

Width="100%"

BorderWidth="1"

Font-Name="Verdana"

Font-Size="8pt"

HeaderStyle-BackColor="#aaaadd"

SelectedItemStyle-BackColor="yellow"

ItemStyle-BackColor="antiquewhite"

AlternatingItemStyle-BackColor="tan"

>

<ItemTemplate>

<asp:HyperLink runat="server"

Text= '<%# Container.DataItem("sitename") %>'

NavigateUrl= '<%# Container.DataItem("siteurl") %>' />

</ltemTemplate>

</asp:DataList>

<i><asp:label id="CacheMsg" runat="server"/></i>

Navigation.aspx is the web form that uses the following component. Navigation.aspx

<%@ Register TagPrefix="Hersh" TagName="nav" Src="nav.ascx" %>

<html>

<head>

<style>

a {color:black;

```
text-decoration:none;}
```

a:hover {color:red;

```
text-decoration:underline;}
```

</style>

</head>

<body>

<form runat=server>

<Hersh:nav id="menu" runat = server

vGridlines = Both

vBorderColor = "Black"

vCellPadding = 7

/>

</form>

</body>

</html>

Most of this code should look familiar because I discussed it in <u>Chapter 6</u>. I have moved the code for populating the data from the <u>page_load</u> event to the <u>LoadData</u> function. The <u>LoadData</u> function first looks at the cache. If it finds data there, it uses it to bind the DataList and displays a message to this effect; otherwise, it reads the XML file to get the data and displays a message telling you that it created the DataSet explicitly. Hence, when you first run the web form that uses this control (navigation.aspx), you should get a message saying, "Dataset created explicitly." If you now press the browser refresh button, the dataset is refreshed from the cache and you get the message, "Dataset retrieved from cache."

I have created a dependency between the cache and the XML file, nav.xml, as follows:

Cache.Insert("MyData", Source, New CacheDependency(Server.MapPath("nav.xml")))

Now whenever the nav.xml file is modified, the DataSet will be re-created. To test this out, modify nav.xml, save it, and refresh the navigation.aspx (don't close and re-run—just press the browser refresh button). You will receive a message that tells you that the DataSet was created explicitly. Please be warned that clicking on the links on the navigation bar will generate an error. This is because we try to navigate to nonexistent web forms.

Summary

The ASP.NET cache can provide substantial performance benefits if used wisely. The manual batch update processes of updating static XML files that are only occasionally modified can now be avoided by setting up a dependency between the XML file and the data cache. This avoids the problems and pitfalls associated with manual intervention.

Chapter 12: Tracing

Overview

When writing code, it is often helpful to output the values of variables for debugging purposes. ASP developers have often resorted to outputting such variable values using Response.Write() statements. The problem with this approach is that these debugging statements need to be cleaned out before putting the application into production. ASP.NET provides a more elegant solution with its tracing services. Instead

of Response.Write(), you use Trace.Write() or Warn.Write() (which outputs messages in red) to write out debugging statements. When you want to put the application in production, you need not delete these debugging statements but just disable Trace for a particular page or for the complete application. ASP.NET provides two levels of tracing services: page-level and application-level tracing. Page-level tracing applies tracing to a single page and is enabled using a Trace = "true" attribute on the top-level Page directive. The Trace output will be shown at the bottom of the page. Application-level tracing is enabled using a "trace" section in the configuration file at the application root directory and this enables trace log output for every page within an application. However, a page-level directive may disable trace for a particular page and that page would be excluded from the trace. The details can be obtained using a utility called trace.axd. The Trace class supports two overloaded methods. These are Trace.Write() and Trace.Warn().These methods are similar except that Trace.Warn() displays output in red. The following are the prototypes for these methods:

Visual Basic.NET

Public Sub [Warn | Write](category As String, message As String, errorInfo As Exception)

End Sub

Public Sub [Warn | Write](category As String, message As String)

End Sub

C#

public void [Warn | Write](String category, String message, Exception errorInfo)

public void [Warn | Write](String category,String message)

For example, you could add the following statement on the page_load event of the page:

Trace.Write("My Trace", "This is Page Load")

The trace output is displayed in a tabular format and the first column of this table is Category and the next Message. Hence, in the above case, My Trace will be displayed under the Category column and This is Page Load under the Message column. It is possible to sort the output table by the Category so that you can see all messages belonging to the My Trace category together.

Page-level Tracing

To enable Tracing for a page, the following directive is added at the top of a page: <%@Page Language="VB" Trace="True"%> You can use the TraceMode attribute to sort by category as follows: <%@Page Language="VB" Trace="True" TraceMode="SortByCategory"%>

To sort by time (the default):

<%@Page Language="VB" Trace="True" TraceMode="SortByTime"%>
Within the body of the page, you would have a number of Trace.Write() or
Trace.Warn() statements. Trace_page.aspx provides an example.
Trace_page.aspx

<%@Page Language="VB" Trace="true" TraceMode="SortByCategory"%>

<html>

<script language="VB" runat="server">

Sub Page_Load(objSource As Object, objArgs As EventArgs)

Trace.Write("My Trace", "This is Page Load")

End Sub

</script>

<body>

Some body matter....

<%

Trace.Write("My Trace", "This is in the Body")

%>

</body>

<%

Trace.Write("My Trace", "OK this is the end..")

%>

</html>

Figure 12.1 shows the output generated.

http://127.8.8.UNSPNieu	d/trace.aups - Microsoft Internet Explorer		
(in gas year (poster	July 100		
HI 2 545 145 2 544	6, THES @ Departon & Share		
	0 0 0 0 0	0 0 7.	koange
phone 🕑 Imp. 1929 Bit Last	Prinsidenes age		
me body matter			
equest Details			
ension fet:	bs3o1bminjp40i45jppro.45	Requ	est Type: CET
ine of Request.	A FRAM TH PL TO HAVE	piets.	s Code: 200
race information	1		
Megory	hereinge	From Field(x)	Prom Lind(4)
6× bola	English the	0.00000	0.000000
p. page	End of the	0.000000	0.000000
by belle	Englis vesencer	0.000000	0.000000
by bolk	end meneraler	0.000000	0.000000
by bela	Degri perestate	0.00000	V-004/006
for byde	End sevestate	3.566363	0.000000
by by Bu	Boges Render	0.000000	0.000000
6× bela	End Kender	0.000000	0.000000
y Trace	Thes is Page Load	0.000000	0.000000
y Trace	This is in the Body	0.000000	0.000000
y Trace	Of this is the end.	0.000000	0.000000
Control Tree			
antrol Id Type	Render Bize bytes (include	ng children) Viewstate Si	re lytes (excluding children)
3408 889.710	e"acts 80	0	
nokies Collection			
10/10	Value		Size
spilessioned	bicas atomings following	gincies.	141
leaders Collection	Value		
	1		in the second

Figure 12.1: Page-level tracing.

Application-level Tracing

I discussed the web.config file in <u>Chapter 10</u>. This file has a Tracing section that is used to set application-wide tracing. For example:

<configuration>

<system.web>

<trace

enabled="true"

pageOutput="false"

requestLimit="20"

traceMode="SortByCategory" />

</system.web>

</configuration>

This section has the four following attributes:

- Enabled = True or False: This setting enables or disables application-wide tracing. Page-level tracing will override this setting for pages where it is set.
- 2. PageOutput = True or False: When set to true, trace output is displayed at the bottom of the page. When set to false, no output is displayed. To see the output, the utility trace.axd is used.
- 3. RequestLimit = some integer: The total number of requests to keep cached in memory on a per-application basis.
- 4. TraceMode = SortByCategory or SortByTime: The user can specify categories when using Trace.Write(). For example, Trace.Write("My Trace", "This is Page Load") can see trace output grouped by categories when the TraceMode is set to be SortByCategory. The SortByTime is the default.

In the samples directory of this folder, I have included a web.config file and a web form called trace_application.aspx to demonstrate application tracing. The web.config file has been described above and trace_application.aspx is the same Web page that I discussed in the context of page-level tracing (trace_page.aspx). The only difference is that there is no page directive at the top of the page; tracing is controlled through the web.config file. Remember that you need to create a virtual path for the folder where you place the web.config file and the trace_application.aspx web form.

First, run the trace_application.aspx with the PageOutput setting set to true. You will see the trace output at the bottom of the file as before. Now set the PageOutput setting to false and run the form. You will note that no output is displayed. To see the trace output you must use trace.axd. This is done by simply request- ing trace.axd in the same application directory that the request for the sample application was made. For example, my application folder is called ASPVirtual and I would request trace.axd by specifying a URL of http://Localhost/ ASPVirtual/trace.axd. Figure 12.2 shows the resultant output. In Figure 12.2 you will note that we are presented with a list of traces to view and we can simply click on a particular trace to view all the details about that trace. The resultant output will be identical to the output obtained from page-level tracing.



Figure 12.2: Using Trace.axd to see the trace output.

Disabling Tracing

When you are ready to deploy the application in production, you should explicitly disable trace.axd by adding the following entry in the HttpHandlers section of config.web:

<HttpHandlers>

<add verb="*" path="trace.axd" type="System.Web.Handlers.TraceHandler" />

<remove verb="*" path="trace.axd"/>

</HttpHandlers >

There is no need to delete the various Trace.Write() statements within the body of the web forms in the application.

Summary

Tracing is a very useful debugging tool. In the past, we had to write a number of Response.Write() statements in the body of a web form to debug problematic code. When the application was ready to be deployed we had to get rid of these statements. This was time-consuming and bug prone because we might delete a line of code with the Response.Write() statements. Tracing avoids these problems because there is no need to remove debugging statements from the web form. You can simply turn off tracing and the debugging statements will not be displayed.

Chapter 13: Security

Overview

Security plays a very important role in Internet applications. Unauthorized users need to be restricted from sensitive portions of the Web sites and authorized users granted access to sections of the site based on their "roles." ASP.NET in conjunction with IIS provides excellent security features.

Security in ASP.NET involves authentication and authorization. Authentication is the process of checking the user credentials (name and password) against authorities called *authentication providers*. If the credentials are verified, the user is considered authenticated. Once authenticated, the authorization process determines whether the user has rights to access the requested resource. ASP.NET can also execute code using the identity of the user. This is known as impersonation. Security is thus a three-step process (the third step—impersonation—is optional):

- User Authentication: This involves verification of the user credentials like name and password and checking against "authentication providers."
- User Authorization: This is the process of determining whether an authorized user has access to the resource requested.
- User Impersonation: This is when the application executes code using the identity of the client making the request.

ASP.NET implements authentication through authentication providers. These authentication providers are modules that contain code required to authenticate the credentials of the requesting user. Three authentication providers are currently available: windows authentication, passport authentication, and forms authentication.

Windows authentication is used in conjunction with IIS authentication. IIS provides authentication using Basic, Digest, or Integrated Windows Authentication. Configuring these authentication options is similar to the way we did it under ASP using the IIS MMC.

Passport authentication is a centralized authentication service provided by Microsoft that offers a single sign-in and core profile services for member sites.

Forms-based authentication service uses cookies to authenticate users and allows an application to do its own credential verification. Unauthenticated requests are directed to an HTML login form. The user supplies his credentials and submits the page. The application authenticates the request against a password repository stored in a database, an XML file, or the web.config file. If the user is authenticated, the system issues a cookie containing the credentials in some form or a key for reacquiring the identity. Subsequent requests are issued with the cookie in the request headers. To activate the authentication services and select the authentication provider, you must configure the <authentication> element in the <security> section of the web.config file. This file was discussed in Chapter 10. Here is a sample security section of a configuration file:

//web.config file

<configuration>

<system.web>

<security>

<authentication mode="[Windows/Forms/Passport/None]">

</authentication>

</security>

</system.web>

</configuration>

After the user is authenticated using any of the authentication providers above, you can further restrict his access based on NTFS's access control list or by permissions on the resources specified in the web.config file. ASP.NET supports role-based security. Users

can be authenticated based on their NT user/group accounts or on custom-defined user/group information specified in either a database or a text file.

Form-Based Authentication

The form-based authentication is the most flexible approach of the three because it allows you to design your own HTML login forms and have more overall control over the authentication process. I will discuss this approach with two examples. The first is a simplified example, which stores usernames and passwords in the web.config file. This will give you a very quick insight into the cookie-based authentication process. The second example is more functional and is relevant to production sites in which I show you how to authenticate passwords against credentials stored in a database table.

A Simple Example of Form-Based Authentication

This example contains three files: web.config, default.aspx, and login.aspx. These files are included in the "simple" folder of the samples folder of this chapter on the book's Web site at <u>www.premierpressbooks.com/downloads.asp</u>. You should create an IIS virtual folder (to mark it as an ASP.NET application) and place these three files there.

The web.config's security section is configured as follows: **web.config**

<configuration>

<system.web>

<authentication mode="Forms">

<forms name=".ASPXUSERDEMO" loginUrl="login.aspx" protection="All" timeout="60">

<credentials passwordFormat="Clear" >

<user name="hersh" password="bhasin"/>

```
<user name="joe" password="smith"/>
```

<user name="test" password="user"/>

</credentials>

</forms>

</authentication>

<authorization>

<deny users="?" />

</authorization>

<globalization requestEncoding="UTF -8" responseEncoding="UTF -8" />

</system.web>

</configuration>

Note that this file has three sections: authentication, authorization, and identity. The meaning of various elements are discussed below:

<authentication mode>

As discussed earlier, authentication mode can be Windows (the default), Forms, Passport, Or None.

<forms>

<forms name=".ASPXUSERDEMO" path = "/" loginUrl="login.aspx" protection="All" timeout="60" />

The name setting gives the name of the cookie, the loginurl specifies the name of the aspx login form, which in this case is login.aspx. The path attribute specifies the path for which the cookie is valid and path = "/" indicates the complete site. The protection attribute can be All, None, Encryption, or Validation. All is the default and uses both encryption and data validation.

<credentials>

<credentials passwordFormat="Clear/SHA1,MD5">:

We store the credentials (meaning usernames and passwords) of the users allowed to access our application in this section. It is not required that we store credentials here, so I will explain how credentials can be stored in a database in the subsequent example. The passwordFormat setting can be Clear, SHA1, or MD5. Clear means that the passwords are stored as clear text and user passwords are compared directly against this value without further transformation. SHA1 and MD5 indicate the hashing algorithms to be used on the passwords. The SHA1 encryption method stores the password in the SHA1 Digest and the MD5 in the MD5 hash digest. Passwords are normally stored in a database. To protect the passwords from prying eyes, they can be encrypted (using SHA1 or MD5 hashing algorithms). This can be done using the HashPasswordForStoringInConfigFile method defined in the FormsAuthentication class. This method accepts the password string and the encryption methods and returns the encrypted password. Here is an example: encrypt.aspx

<%@ Import Namespace="System.Data" %>

<html>

<head>

<script language="VB" runat="server">

Sub Page_Load(Src As Object, E As EventArgs)

Dim SHA1Password As string

Dim MD5Password As string

SHA1Password = FormsAuthentication.HashPasswordForStoringInConfigFile ("Hersh","SHA1")

MD5Password = FormsAuthentication.HashPasswordForStoringInConfigFile ("Hersh","MD5")

response.write("Password in SHA1 Format : " + SHA1Password + "
>")

response.write("Password in MD5 Format : " + MD5Password)

End Sub

</script>

</head>

</html>

In this example, I have encrypted the username ${\tt Hersh}$ using the two methods. Here is the result:

Password in SHA1 Format: BBB0E8F21D0747A3A4927477F86886F1AE6FBE78

Password in MD5 Format: 52F79F51B8A443BC6F915A585BB0C1FF Typically, you will store the encrypted password in the database or the web.config file. The .HashPasswordForStoringInConfigFile method can only encrypt passwords and cannot be used to decrypt it. You can use the .NET cryptography APIs available at System.Security.Cryptography namespaces to encrypt and decrypt the data. DevPower Solutions offer a free version of the .NET cryptography component at http://www.devpower.com/Encrypt.NET.

The username and password subsection specifies the legal users of the application and their passwords.

<authorization>

The authorization section has an <allow> and a <deny> section that you can use to allow or deny user access. For example, to deny access to users with anonymous identity you can use the special identity "?" as follows:

<deny users="?" />.

To allow access to all users, you can use the "*" identity as follows:

<allow users = "*"/>

The loginurl setting of the cookie subsection specifies the login form to use, which is the form <u>login.aspx</u>. This form has the following code:

```
Login.aspx
```

<%@ Import Namespace="System.Web.Security " %>

<html>

<script language="VB" runat=server>

Sub Login_Click(Src As Object, E As EventArgs)

If FormsAuthentication.Authenticate(UserName.Value, UserPass.Value)

FormsAuthentication.RedirectFromLoginPage(UserName.Value,PersistCookie.Checked)

Else

```
Msg.Text = "Sorry Invalid username or password : Please try again"
```

End If

End Sub

</script>

<body>

<form runat=server>

```
<h3><font face="Verdana">Login Page</font></h3>
```

User Name:

<input id="UserName" type="text" runat=server/>

runat=server/>

Password:

<input id="UserPass" type=password runat=server/>

runat=server/>

Persistent Cookie:

<ASP:CheckBox id=PersistCookie runat="server" />

<asp:button text="Login" OnClick="Login_Click" runat=server/>

<asp:Label id="Msg" ForeColor="red" Font-Name="Verdana" Font-Size="10" runat=server />

</form>

</body>

</html>

The body of this form has two textboxes: UserName and UserPassword and one checkbox: PersistCookie. When this checkbox is checked, the cookie is persisted; that is, the cookie information is not lost when the browser is closed. When the user clicks on the login button, the login_click event is fired. This event first uses the authenticate method of the FormsAuthentication class to authenticate the user credentials. If the user is authenticated, the RedirectFromLoginPage method is used to store the cookie (and if the PersistCookie checkbox is checked, to persist the cookie) and to redirect the user back to the page that was actually requested by the user.

The default.aspx form is the form that the user originally requests, and after the login process, gets to see. This is its listing:

default.aspx

<%@ Import Namespace="System.Web.Security " %>

<html>

<script language="VB" runat=server>

Sub Page_Load(Src As Object, E As EventArgs)

msg.Text = "Welcome, " + User.Identity.Name

End Sub

Sub Signout_Click(Src As Object, E As EventArgs)

FormsAuthentication.SignOut()

Response.Redirect("login.aspx")

End Sub

```
</script>
```

<body>

<h3>Using Cookie Authentication</h3>

<form runat=server>

<h3><asp:label id="msg" runat=server/></h3>

<asp:button text="Signout" OnClick="Signout_Click" runat=server/>

</form>

</body>

</html>

In the page load event of this page, I access the user's name using the $\tt User$ class as follows:

msg.Text = "Welcome, " + User.Identity.Name

The User class is available intrinsically from a Web page and has three methods: IsAuthenticated, Name, and Type. The IsAuthenticated method returns a Boolean specifying whether the user has been authenticated. The Name method returns the username, and the Type method returns the type of authentication used. Finally, clicking on the signout button clears the cookie using the SignOut method of the FormsAuthentication class as follows:

Sub Signout_Click(Src As Object, E As EventArgs)

CookieAuthentication.SignOut()

Response.Redirect("login.aspx")

End Sub

Figure 13.1 is the screenshot of the login page. Figure 13.2 is the screenshot of a successful login.



Figure 13.1: The login page.



Figure 13.2: A successful login.

Using a Database to Store Passwords

In this example, I will show you how you can use form-based authentication with the passwords validated against a database table. I will store the passwords in an access database called security.mdb. This implementation will comprise of three files: web.config, default.aspx, and login.aspx. These files can be found in the "advanced" folder of the samples folder of this chapter on the book's Web site at <u>www.premierpressbooks.com/downloads.asp</u>. The web.config and default.aspx files are identical to those discussed in the earlier example, with the exception that we do not store the user credentials in the web.config file. The only file that is different is the login.aspx file, the listing of which is below:

```
login.aspx (database version)
```

<%@ Import Namespace="System.Data" %>

```
<%@ Import Namespace="System.Data.OleDb" %>
```

<%@ Import Namespace="System.Web.Security " %>

```
<html>
```

<script language="VB" runat=server>

Sub Login_Click(Src As Object, E As EventArgs)

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Dim dv As DataView

ConnStr = "PROVIDER=Microsoft.Jet.OLEDB.4.0;DATA SOURCE=" & server.mappath("security.mdb")

myConnection = New OleDbConnection(ConnStr)

'DataSetCommand

SQL = "select * from passwords"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "passwords")

dv = new DataView(ds.Tables("passwords"))

dv.Sort = "UserName"

dv.RowFilter = "UserName = '" + UserName.Value + "' and password = '" + UserPass.Value +""

if dv.count >=1 then

FormsAuthentication.RedirectFromLoginPage(UserName.Value, PersistCookie.Checked)

Else

Msg.Text = "Sorry Invalid username or password : Please try again"

End if

End Sub

</script>

<body>

<form runat=server>

<h3>Login Page</h3>

User Name:

<input id="UserName" type="text" runat=server/>

runat=server/>

Password:

<input id="UserPass" type=password runat=server/>

runat=server/>

Persistent Cookie:

<ASP:CheckBox id=PersistCookie runat="server" />

<asp:button text="Login" OnClick="Login_Click" runat=server/>

```
<asp:Label id="Msg" ForeColor="red" Font-Name="Verdana" Font-Size="10" runat=server />
```

</form>

</body>

</html>

In this example, I populate a DataSet with rows from the passwords table of security.mdb. I then assign the default view of this DataSet to a DataView and filter for the username and password supplied by the user as follows:

dv.Sort = "UserName"

```
dv.RowFilter = "UserName = '" + UserName.Value + "' and password = '" + UserPass.Value +"'"
```

Finally, I check for the number of rows returned by using the count property of the

DataView. If the count > 1 then I log the user in; otherwise, I reject his credentials as follows:

if dv.count >=1 then

 $\label{eq:starsest} Forms Authentication. Redirect From Login Page (User Name. Value, Persist Cookie. Checked)$

Else

Msg.Text = "Sorry Invalid username or password : Please try again"

Passport Authentication Provider

Passport Authentication is a centralized authentication service provided by Microsoft offering a single sign-in and core profile services for member sites. If you have an account on Hotmail, passport is being used to authenticate you. At the time of this writing, Microsoft had just released information and a White Paper on Hailstorm. This product is a major technology component of Microsoft's .NET vision and will include Web versions of Hotmail, MSN Messenger, and the Passport user authentication product. The gist is that Passport will be like an Internet passport. Your critical personal information will be stored at a central location and you will have full control over it. Hence you will use the same password and username over all partner sites. Your personal information will not have to be re-keyed in at these sites. The same would apply to your address book and your bookmarks. These would be accessible from any computer over the Web, and would be created only once. I have discussed various issues pertaining to Hailstorm in <u>Appendix B</u>.

The general process of implementing the Passport authentication service is as follows:

- Download and install Passport SDK from <u>http://www.passport.com/business</u>. You will need to register and *pay* fees for the SDK.
- 2. Set up Passport as the authentication mode in the configuration file as follows:
 - 3. <configuration>
 - 4. <system.web>
 - 5. <security>
 - 6. <authentication mode ="Passport">
 - 7. </authentication>
 - 8. </security>
 - 9. </system.web>

</configuration>

10. Implement Passport authentication and authorization following the directions in the Passport documentation.

Passport requires that users of a site be registered with Passport. Users can register by creating Hotmail or MSN accounts or by going directly to the password site and registering there. When a user requests a protected resource and the request does not contain a valid Passport ticket (cookie), the server returns a 302 error message and redirects the user to the Passport Login service. Encrypted parameters about the original request are passed on to Passport with the query string. Passport presents a login form to the user, who supplies the required credentials and does a postback using SSL (Secure Socket Layers). The login server of Passport authenticates the user and redirects him back to the original URI with the authentication ticket encrypted in the query string. The originating server detects the absence of the cookie and the presence of the ticket on the query string and issues an authentication cookie. Subsequent requests for the resource at the site are authenticated using the supplied ticket.

Windows-Based Authentication

Windows - or IIS -based authentication is straightforward. You just enable IIS basic authentication in the IIS MMC, set up the users allowed to access a particular Web application, and set up windows as the authentication mode in the configuration file.

End if

Enabling Basic Security Authentication

The following example is using Windows NT.

- 1. Open IIS MMC. Right-click on Default Web Site and select Properties/Directory Security/Edit.
- 2. Check the Basic Authentication checkbox, as shown in Figure 13.3.



Figure 13.3: Enabling Basic security.

Setting Permissions

You now need to set up ACL (access control lists) based on users or groups for your application.

1. Go to the *physical folder* (not virtual folder) of the application. Rightclick and select the Security tab. On Windows NT you will see the screen shown in Figure 13.4.

Name		Add
Everyone 2	- 1	Bemove
		2
emissions:	Allow	Deny
Full Control		묘
Modily Band & Emande		H
List Folder Contents		H
Read		H
Write		-

Figure 13.4: Setting permissions in Windows NT.

2. Set up the ACL based on users or groups for your application.

Edit the Web.config File

Modify the security section of the web.config file to enable Windows authentication as follows:

<configuration>

<system.web>

<security>

<authentication mode ="Windows">

</authentication>

</security>

</system.web> </configuration>

When a user tries to access a resource in the application, he is asked to log in, as shown in Figure 13.5.

20	Please type y	our user name and password.
IJ	Site:	127.0.0.1
	User Name	
	Password	
	Domain	
	E Save this	password in your password list

Figure 13.5: Windows-based login.

Summary

ASP.NET, combined with IIS, provides commendable security features. The form-based security features provide the greatest amount of flexibility and the examples provided in this chapter can be successfully applied to production sites. The Passport authentication provider still needs to be tested for user acceptance. At the time of this writing, companies like American Express, Click Commerce, eBay, Expedia.com, and Groove Networks had adopted this provider to meet their authentication needs. Given Microsoft's vision of Passport playing a pivotal part in the .NET design, many more companies are sure to follow. I further discuss Passport, together with Hailstorm, in <u>Appendix B</u>.

Part II: Projects

Project List

Project 1: A Personal Finance Manager Project 2: Web Services Project 3: Inventory Management System Project 4: The GenEditAdd Control Project 5: Visual Studio.NET

Project 1: A Personal Finance Manager

Chapter List

- <u>Chapter 14:</u> The Design of the Personal Finance Manager
- Chapter 15: Chart of Accounts

<u>Chapter 16:</u> Transactions

Chapter 17: The Trial Balance Report

Project 1 Overview

A Personal Finance Manager is an accounting application like Quicken or Microsoft Money that enables you to maintain bank accounts, cash, credit cards, and investment accounts. You record all your deposits and expenses and the system generates many reports that help you to monitor your financial health.

An accounting application is composed of many modules, each performing certain specific tasks. A Personal Finance Manager handles cash, bank accounts, credit cards, and other financial transactions. An Inventory module records movement of inventory items, an Accounts Receivable module records customer transactions, and an Accounts Payable module records supplier transactions. These modules need to talk to each other. For example, when a customer buys some goods, the Financial Accounting module needs to reflect the monetary value of the purchase, and the Inventory module needs to be updated with the physical movement of the inventory items.

Traditionally, accounting applications have been developed around the client/server model. The modules share a central database and are connected to each other with some sort of a network protocol.

The Internet brings some very exciting possibilities to the traditional way of designing applications. The various modules of an accounting application need no longer be connected with wire. Using ASP.NET and web services, we can design applications that can send and receive data using the Internet and HTTP.

In this Project, I want to show you how to build a Web-Enabled Personal Finance Manager using ASP.NET web forms (we will extend this to incorporate web services later in the book). Its design is steeped in the double entry system of accounting. You can easily extend the concepts developed in this chapter to build other accounting modules. In fact, each new module is actually a new web form that will be very similar to the one that we design here.

Chapter 14: The Design of the Personal Finance Manager

The Personal Finance Manager is based on the double entry system of accounting. I will explain the relevant theory as I go along. In order to run the application, you need to create a Microsoft SQL Server database called ASPNET and install various database objects in it. <u>Appendix A</u>, "<u>Installing the Sample Database</u>," outlines the steps required to accomplish this. The Personal Finance Manager uses five database tables. These are the <u>groups</u>, masters, tr_header, transactions, and the tblSelection tables. An update, delete, and insert trigger is associated with the transactions table. This application also makes use of two stored procedures. These are the p_masters and the p_transactions stored procedures.

Groups

There are four basic groups and two types in financial accounting. The groups are Assets, Liabilities, Income, and Expenses, and the two types are Debits and Credits. Each group is either of type "Debit" or type "Credit." Assets and Expenses are Debit types, and Income and Liabilities are Credit types. <u>Table 14.1</u> outlines the relationship between Types and Groups.

Table 14.1 Relationship between Types and Groups

Group	Debit Type	Credit Type
Asset	x	
Liability		x
Income		x
Expense	x	

Financial transactions are classified under these groups (or subgroups under these groups). This allows you to prepare a Balance Sheet, a Profit and Loss statement, or a cash flow statement from your financial data. The purpose of each of these reports is as follows:

- A Trial Balance is a report that shows all debit accounts on one side and all credit accounts on the other. The sum of all credit balances should always match the sum of all debit balances. The Trial Balance is the basis of preparing the Profit and Loss account and the Balance Sheet.
- Subtracting all Expenses from Income derives a Profit or Loss figure. Thus: Profit or (Loss) = Income - Expenses
- The Balance Sheet is a report that shows Assets plus Inventory on one side and Liabilities + Profit or Loss (as derived above) on the other. These two sides should be equal.

The Groups Table

I have provided some predefined groups that you can use in defining your Chart of Accounts. Each Master Account in your Chart of Accounts will specify a group to which it belongs. When you want to prepare a report like Balance Sheet or Profit and Loss, you will do a summation based on groups. Thus, if you want to find out the total expenditure, you would write a query as follows:

"Select sum (closing) from masters where code_category = '700'".

Here '700' is the code_category ("Group") for expenditure accounts. You can use the <u>groups</u> table as it is. <u>Table 14.2</u> is the definition of the <u>groups</u> table.

Table 14.2 The Groups Table Definition

Column	Туре	Length	Description
id	char	3	"G" (group) or "R" (reserved group)
code_display	char	30	Descriptive Name
code_value	integer		Primary Value
code_category	integer		Parent Group
type	char	1	"A","L", "I", "E"

Table 14.3 lists out the predefined groups included in the groups table.

Table 14.3 The Predefined Groups					
id	code_display	code_value	code_category	type	

Table 14.3 The Predefined Groups				
id	code_display	code_value	code_category	type
R	Capital account	1		L
R	Long term loans	2		L
R	Current liabilities	3		L
R	Fixed assets	4		A
R	Investments	5		A
R	Current assets	6		A
R	Revenue accounts	7		М
R	Branch/divisions	8		А
R	Reserves and surplus	9		L
R	Secured loans	201	2	L
R	Unsecured loans	202	2	L
R	Duties taxes payable (bs)	300	3	L
R	Provisions	301	3	L
R	Sundry creditors	302	3	L
R	Bank od & limits	303	3	L
R	Deposits (assets)	601	6	A
R	Advances (assets)	602	6	A
R	Sundry debtors	603	6	A
R	Cash in hand	604	6	A
R	Bank accounts	605	6	A
R	Sales account	700	7	I
R	Purchase account	701	7	E
R	Income account	702	7	I
R	Duties and taxes paid (p/l)	703	7	E
R	Expenditure account	704	7	E
R	Advances - excise a/cs	60200	602	A
R	Expenses (direct)	70400	704	E

Table 14.3 The Predefined Groups					
id	code_display	code_value	code_category	type	
R	Expenses (indirect)	70401	704	E	

Note the coding structure of this table. The primary groups have a code_value (the primary key value) of 1 to 9. Each subgroup under the primary group has a code_value of 100 multiplied by the code_value of the parent group. Each sub-group account in the same level has a sequential code value. Take the Revenue account classification. Revenue account, which is the base account group for all Profit and Loss items, has a code_value of 7. The Sale, Purchase, Income, Duties & Taxes paid and Expenditure accounts all fall under the Revenue account classification. Hence they have code_values of 700,701,702,703,704 respectively. Again the Expenses (direct) account has a code_value of 70400. This is because it falls under the Expenditure group which has a code_value of 704.

This method of code classification allows us to build queries that get all records for a given group or subgroup. As I will tell you in a moment, each master account in the Chart of Accounts is associated with a group. Each master account has a closing balance field that is updated each time a transaction takes place. Suppose we wanted to find the sum of the closing balance for all revenue accounts. The SQL query for this is: "Select sum(closing) from masters where code_category = 7". Now suppose we wanted to get all the expenditure accounts. The SQL query for this would be: Select * from masters where substring(convert(varchar(13),code_value),1,3) = '704'. If we apply this same query against the groups table we will get Expenditure account, Expenses (direct) and Expenses (indirect). Similarly to get only the Expenses (direct) records, we would say Select * from masters where substring(convert(varchar(13),code_value),1,5) = '70400'.

The Masters Table

A master account must be created for each Income, Expense, Asset, and Liability account that you want to use. This is called the Chart of Accounts. The masters table holds this information. Each master account must be associated with a group from the <u>groups</u> table. You specify the group in the code_category field of the masters table. Each account has a closing field. This field holds the closing balance of that account at any given time. This field is automatically updated by triggers on the transactions table. You use this field for displaying the closing balances in the Trial Balance, Profit & Loss, and the Balance Sheet. <u>Table 14.4</u> is the definition of the masters table.

1

Column	Туре	Length	Description	
code_value	integer		ldentity, Primary Value	
code_display	char	30	Descriptive Name	
code_category	integer		Group	
type	char	1	A, L, I, E	
closing	money		Closing Balance	
opening	money		Opening	

Table 14.4 The Masters Table

Table 14.4 The Masters Table				
Column	Туре	Length	Description	
			Balance	

The Transactions Header Table

The tr_header table records the header information for the transactions. This is information like date, narration, document number, and so on. The primary key of the tr_header is the document number (doc_no). Table 14.5 is the definition of the tr_header table.

Table 14.5 The tr_header Table

Column	Туре	Length	Description
ld	char	3	Voucher Type ("Bank", "Sales", "Purchases")
date	datetime		Voucher Date
doc_no	int		Primary Key
narr	varchar	150	Narration
ref	varchar	15	Reference

The Transactions Table

This transaction information will be stored in the transactions table. There will be a debit and a credit transaction entry for each deposit or withdrawal. The primary key of the transactions table is the document number and the serial number ($doc_no + sn$). Table 14.6 gives the description of the transactions table.

Table 14.6 The Transactions Table				
Column	Туре	Length	Description	
doc_no	integer		Primary Key	
sr_no	money		Primary Key	
code_value	integer	n/a	First Masters a/c Debited or Credited	
dr_amount	money	n/a	Debit Amount	
cr_amount	money	n/a	Credit Account	
posted_to	integer	n/a	Second Masters a/c Debited or Credited	

The tr_header and the transactions tables are related on the doc_no field in each table. There exists a one-to-many relationship between the two tables.

TblSelection Table

The tblSelection table has a structure as shown in Table 14.7.

Table 14.7 The tblSelection Table			
Column	Туре	Length	Description
Selection	Varchar	50	Records user account selection

This table has a single column called selection. This table is used with the Transactions web form that will be developed in Chapter 16. It is used to record the account selection made by a user.

Figure 14.1 shows the database schema for this application. Ter: P 30 P ĩe

Figure 14.1: The database schema for the Personal Finance Manager application.

Chapter 15: Chart of Accounts

In this chapter, I will build a web form that will be responsible for maintaining the masters table. The maintenance functionality will include procedures to list, add, modify, and delete master records.

Inserting and Updating Master Records

The logic for the insertion or deletion of records from the masters table is encapsulated in the stored procedure p_masters. All the fields of the masters table are passed to this procedure as input parameters. I introduced the p masters stored procedure in Chapter 4, "Data Binding," and showed how it was called from either a DataGrid or a DataList.

The code_value is the primary key of the masters table. If a null code_value is passed to the procedure, it builds an insert SQL action query using the passed parameters. Otherwise it updates the masters table record which has the same code_value as the one passed to the stored procedure.

In the case of a new record creation, the stored procedure assigns an account type to the record. As explained in Chapter 14, "The Design of the Personal Finance Manager," this account type can be A, L, I, or E. This account type is determined by looking up the type column in the groups table. To look up this value, you can pass either the code of the group (the @code_category parameter) or the descriptive name of the group (the @group_name parameter) to the stored procedure. If the @group_name parameter is provided the group details are looked up as follows:
SELECT @grtype = type ,

@grCode_value = code_value

from Groups

WHERE code_display = rtrim(@group_name) If this parameter is not supplied, the code_category parameter should be supplied and this is used to look up the group details as follows:

SELECT @grtype = type ,

@grCode_value = code_value

from Groups

WHERE code_value = @code_category The following is the complete listing of p_masters:

Stored Procedure p_masters

create procedure p_masters

@code_value integer = null,

@code_display varchar(30),

@code_category integer = NULL ,

@type char(1)= NULL,

@opening money = 0,

@closing money =0,

@group_name varchar(30) = NULL

as

This procedures creates or updates a new master record. If a null

code_value is passed, a record is inserted else the record is updated. You can

pass either the code of the group or the descriptive name of the group.

Example passing the code of the group (604):

execute p_masters 1,' Petty Cash a/c' ,604, 'A',0,0 ,NULL

Example passing the name of the group(Cash a/c):

p_masters 1,'Petty Cash a/c' ,604, 'A',0,0,'Cash a/c'

DECLARE @flag integer

DECLARE @oldType as char(1)

DECLARE @grCode_value integer

DECLARE @grType as char(1)

IF isnull(@code_value,0) = 0

-----If code value = 0 then INSERT a new record ------

BEGIN

IF Datalength(@group_name) > 1

---- If the group_name is provided look up group

-----details using the descriptive name

Begin

--Get Group Details

SELECT @grtype = type ,

@grCode_value = code_value

from Groups

WHERE code_display = rtrim(@group_name)

End

Else

--- If a numeric code_value of the group is provided,

```
---look up the group details using it
```

Begin

```
SELECT @grtype = type ,
```

@grCode_value = code_value

from Groups

WHERE code_value = @code_category

End

Insert into masters(code_category,code_display,type,opening,closing)

 $Values (@grCode_value \ , @code_display, @grtype, \\$

isnull(@opening,0),isnull(@closing,0))

IF @@ERROR != 0

Begin

GOTO doerror

End

END

ELSE

-----UPDATE a record if a code_value is passed------

BEGIN

Update masters

Set code_category = @code_category,

code_display = @code_display,

--type = @type, don't allow update of type

opening =@opening,

closing =@closing

Where code_value =@code_value

IF @@ERROR != 0

Begin

GOTO doerror

End

END

SELECT 0

GOTO doreturn

doerror:

Return - 100

doreturn:

RETURN 0

GO

The Masters Web Form

The Masters web form is the form that adds, updates, and deletes rows from the masters table. In <u>Chapter 4</u>, I built the Masters form, step by step, and my last step was

to build Masters2.aspx. I built a call to the stored procedure p_masters but did not actually execute it. Instead I wrote out the procedure call syntax to the screen. Masters2.aspx did not have any addition or deletion capabilities. In this section, I will enhance this form so that I can add, delete, and update records to the masters table. Figure 15.1 shows what the form looks like.



Figure 15.1: The Masters web form.

<u>Figure 15.2</u> shows what it looks like in Add mode. <u>Figure 15.3</u> shows what it looks like in Edit mode.

27 A (3 >568 • -	1		T							
⇒task • ⇔	121		Hearth + LT	* YI DO	okraeks	attempting to o	ormentito Vishoot		Tie 3	10
	.01	000	Search (1) Fo	VORTEN	CHERT	· 6.0	CRI .			
ochress 😰 hrs	rij/bcaho	ts'AspAintBool	(Overlay Schurz	3_samp	ies,Mesters	stands.			*	e2 50
nks @)Caston	ite Linis	() THE HO	nal @jseach	Share	Subnesson	istes igan	nciovis Media 🐑 W	indons		
										1
hart of	Acco	unts								
	Account	No. of Concession, Name	-				Acro Account	1		
Less Dellese	•	Name	Greep	171	He opens	ng Lionny	Add a New Acc	ount:		
Edit Celeta	1.9	Farpo	eccounts .	Α.	0	13	Name:		_	
Lot Celete	19	Salary	Inceme account	1	0	29	Grant D	freeze lation	-	
tot Celeta	22	Utilties	Expenditure		0	13	Opening Values	1		
Lost Dalata	-	Wate	bank.				obsend rear b	<u>.</u>		
La Latera	2.0	Deterer	accounts back	~			Submit			
cit Deleta	27	Card	eccounts	A	. 0	0				
ante Mise	15	2• ⊤⊦	A Mag	eto	re w	eh for	m in the	a add mo	cuculanturet	0.000
igure	15.	2: Tł	ie Mas	ste	rs w	eb for	m in the	e add mo	ode all	ows
igure	15.	2: Th	ne Mas	ste	rs w	eb for	m in the	e add mo	ode all	ows
gure	15.	2: Th		ste	rs w		rm in the	e add mo	ode all	ows
bee igure Tradite 27 A (1) bask -		2: Th		ste	IS W		rm in the	e add mo	ode all	ows • • •
inne igure inne ben inne ben inne ben inne ben inne ben inne ben inne ben inne inne inne inne inne inne inne i		2: Th	ne Mas serie (1) serie (1)	Ste Ste State State State	rs w Grants Grants subreson	eb for Menuou la co v [Co e co st. accv stees @ver	rm in the arrestic resolution consider and a consider	e add mo eterr	ode all	ows
are gure tradient dess (2) ers s (2) cases hart of		2: Th	ne Mas Start (the st earth - Ut Starth (the st Charter (the start (the start)	Ste son D son D son D son D	IS W		rm in the arrest of the contract arrest of the contract arrest of the contract arrest of the contract of the contract of the contract arrest of the contract of the cont	e add mo alarer	ode all	ows
ave gure toological bas toos glosses hart of at		2: Th	ne Mas ster2schus) (it seen @re ne @beed ne @beed	Ste Ste State Superior	rs w Grada	eb for atoget to r 20 a to stage stage stage	rm in the result iteration methy failed.	e add mo eliver ndors	ode all	ows e
ave gure lood set these of these of the these of these of these of these of these of these of these of		2: Th	ne Mas sterðikhað í serði (*) (*) jesen (*) (*) jónetriðnur af (*) jesent sterga	Ste engl - nite variation i biogram	rs w Marana Marana Subresson Subresson Dank ac	eb for not appresent	Tim in the second bit care is a create Value.	e add mo alarer Indons Operation	ode all	ows
are gure to A are bas e to A e to	15.	2: Th	ne Mas stradiches) seen () (1 seen () seen () seen () seen () seen s fans s	Ste "Indu Varter Dame	rs w constants const	eb for Educate Ma attracting to O v 2000 2000 2000 concrete accounts	rm in the case between the meeter reduced cover reduced and the reduced and the reduced and th	e add mo elarer micres	Conservation ode all ne 1	
are gure logical basis tass () ten tess () ten tess () tess tess () tess () tess tess () tess () tess tess () tess () tess tess () tess () tess () tess tess () tess	15.	2: Th 2: Th 2: Charles 2: C	ne Mas stratychoste seetu e (17 seetu e (27 seetu e (27 seetu e (27 seetu e (27) seetu e (27) se	Ste Ste State	IS W Constants Constants Subresson Constants Const	eb for ratempte to r 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	Type A A A A A A A A A A A A A A	e add mo elerer ntors		OWS P P P P P P P P P P P P P

Figure 15.3: The Masters web form in the edit mode allows you to modify existing records.

I explained most of this script in <u>Chapter 4</u> while discussing Masters2.aspx, so I will only describe the additions to this form here.

Update Logic

The Sub Grid1_Update handles the Update logic. This function is fired when the grid is activated in the Edit mode. In <u>Chapter 4</u>, I discussed building the SQL action query string that makes a call to the stored procedure p_masters.

Grid1_Update Sub Grid1_Update(sender As Object, e As DataGridCommandEventArgs) Dim sql As string Dim code_display As String Dim code_category As String Dim type As String Dim opening As String Dim closing As String Dim myTextBox As TextBox 'This is the key value : Retrieved from the DataKey, since it's a read only field Dim code_value as string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString myTextBox = E.Item.FindControl("edit_name") code display = mytextbox.text myTextBox = E.Item.FindControl("edit_group") code_category = mytextbox.text myTextBox = E.Item.FindControl("edit_type") type = mytextbox.text myTextBox = E.Item.FindControl("edit_opening") opening = mytextbox.text myTextBox = E.Item.FindControl("edit_closing") closing = mytextbox.text 'Now execute stored procedure sql = "Execute p_masters " + code_value + ", '" + code_display + " ',"

sql = sql + code_category + ", '" + type +"' ," + opening + "," + closing

RunSql(sql)

End Sub

This SQL query string is passed on to the function RunSql that does the actual work of executing the SQL statement. Note that I extract the primary key (code_value) and pass it on to the procedure. The existence of a valid code_value tells the procedure to issue an update statement. If you pass it a null code_value, it will issue an insert statement.

Adding Records

Three textboxes and one button have been added to the form. These controls reside on a panel that has an id of AddPanel. In the aspx form, I have added HTML comments to show where the section begins and ends.

Insert Logic Is the Form

<! ----- insert row logic----->

```
<asp:Panel id="AddPanel" runat="server" Visible="false">
```

Add a New Account:

```
Name:
```

<asp:TextBox id="acode_display" runat="server" />

```
<asp:RequiredFieldValidator runat="server"
```

controltovalidate=acode_display

```
errormessage="Name is required.">*
```

```
</asp:RequiredFieldValidator>
```

Group:

```
<asp:DropDownList DataTextField = "code_display"
    DataValueField = "code_value" id="acode_category" runat="server" />
  Opening Value: 
  <asp:TextBox id="aopening" value = "0" runat="server" />
  <asp:Button id="SubmitDetailsBtn" text="Submit"
    onclick="add Click" runat="server" />
  </asp:Panel>
```

```
<! ------Insert Logic ends----->
```

A button having an id of AddShow displays a caption "Add Account" on the web form. Clicking this button fires the add_show Sub. This Sub simply sets the visible property of the panel to true. When the panel is visible, all the controls on the panel also become visible. At this point, all the textboxes are ready for accepting user input. The insert logic is handled by the function add_click. It builds a SQL query string by extracting the text properties of various textboxes. The following is the Sub:

The add_click Sub

Sub add_click(Source As Object, E As EventArgs)

Dim sql As string

```
If acode_display.text = "" or acode_category.SelectedItem.Text = "" then
```

response.write("Incomplete information")

exit sub

end if

```
SQL = "Execute p_masters @code_value=NULL,"

SQL = SQL + " @code_display = '" + acode_display.text + "', @group_name = '"

SQL = SQL + acode_category.SelectedItem.Text + "', @type = NULL,"

SQL = SQL + " @opening =" + aopening.text + ", @closing = 0"

RunSql(sql)

'reset values

acode_display.text = ""

aopening.text = ""

hidePanel()
```

End Sub

Note that we are passing a NULL code_value to the procedure. This fires an insert statement. This SQL query string is passed on to the function RunSql, which does the actual work of executing the SQL statement.

Delete Mode

The Delete mode is activated when the user clicks on the delete link. I simply build a delete SQL action query and pass it on to the function RunSql that actually executes this query.

Sub Grid1_delete

Sub Grid1_delete(sender As Object, e As DataGridCommandEventArgs)

Dim code_value As string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString

Dim sql As string

sql = "Delete from masters where code_value = " + cstr(code_value)

RunSql(sql)

End Sub

The RunSql Function

This is a generic function that can execute a SQL action query. The SQL query is passed to it as a parameter. The <u>Grid1_update</u> Sub, the Add_click Sub, and the Grid1_delete Sub make use of this function to update, add, or delete a record. This function opens a database connection, executes the query using ExecuteNonQuery, and finally closes the connection. It catches OleDbExceptions and other exceptions and writes them out to the screen in red.

```
Sub RunSql
```

Sub RunSql(sql as string)

try

Dim mycommand2 As New OleDbCommand(sql,myConnection)

myConnection.Open()

myCommand2.ExecuteNonQuery()

myConnection.Close()

'turn off editing

Grid1.EditItemIndex = -1

Catch ex As OleDbException

' SQL error

Dim errItem As OleDbError

Dim errString As String

For Each errItem In ex.Errors

errString += ex.Message + "
>"

Next

Message.Text = "SQL Error.Details follow:
>/>/>" & errString

Message.Style("color") = "red"

Catch myException as Exception

Response.Write("Exception: " + myException.ToString())

Message.Style("color") = "red"

End try

rebind

response.write(sql)

End Sub

The following is the complete code listing: Masters3.aspx

<%@Page Language="VB" Inherits="BaseClass" Src="masters3.vb" %> <%@ Import Namespace="System.Data" %> <%@ Import Namespace="System.Data" %> <%@ Import Namespace="System.Data.OleDb" %> <html> <script language="VB" runat="server"> Sub Grid1_Update(sender As Object, e As DataGridCommandEventArgs) Dim sql As string Dim code_display As String Dim code_category As String Dim type As String Dim opening As String Dim closing As String Dim myTextBox As TextBox 'This is the key value : Retrieved from the DataKey, since it's a read only field Dim code_value as string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString myTextBox = E.Item.FindControl("edit_name") code display = mytextbox.text myTextBox = E.Item.FindControl("edit_group") code_category = mytextbox.text myTextBox = E.Item.FindControl("edit_type") type = mytextbox.text myTextBox = E.Item.FindControl("edit_opening") opening = mytextbox.text myTextBox = E.Item.FindControl("edit_closing") closing = mytextbox.text 'Now execute stored procedure sql = "Execute p_masters " + code_value + ", "" + code_display + "'," sql = sql + code_category + ", " + type +"'," + opening + "," + closing RunSql(sql)

```
End Sub
 Sub add_click(Source As Object, E As EventArgs)
  Dim sql As string
  If acode_display.text = "" or acode_category.SelectedItem.Text = "" then
   response.write("Incomplete information")
   exit sub
  end if
  SQL = "Execute p_masters @code_value=NULL,"
  SQL = SQL + " @code_display = "" + acode_display.text + "', @group_name = ""
  SQL = SQL + acode_category.SelectedItem.Text + "', @type = NULL,"
  SQL = SQL + " @opening =" + aopening.text + ", @closing = 0"
  RunSql(sql)
  'reset values
  acode_display.text = ""
  aopening.text = ""
   hidePanel()
 End Sub
 Sub add_show(Source As Object, E As EventArgs)
  AddPanel.visible = true
 End Sub
 Sub Grid1_delete(sender As Object, e As DataGridCommandEventArgs)
  Dim code_value as string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString
  Dim sql As string
  sql = "Delete from masters where code_value = " + cstr(code_value)
  RunSql(sql)
 End Sub
</script>
<body style="font: 10pt verdana">
```

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

<asp:ValidationSummary runat=server headertext="There were errors on the page:" />

<asp:HyperLink runat="server" Text="Trial Balance" NavigateUrl="TrialBalance.aspx"> </asp:HyperLink> <asp:HyperLink runat="server" Text="Transactions" NavigateUrl="selection.aspx"> </asp:HyperLink> <asp:HyperLink> <asp:HyperLink runat="server" Text="Home" NavigateUrl="default.aspx"> </asp:HyperLink> <asp:HyperLink runat="server" Text="Home" NavigateUrl="default.aspx"> </asp:HyperLink> </asp:HyperLink> <h3>Chart of Accounts </h3> <asp:Label id="Message" runat="server"/> <asp:Button id="Addshow" text="Add Account" onclick="add_show" runat="server" />

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

OnEditCommand="Grid1_Edit"

OnCancelCommand="Grid1_Cancel"

OnUpdateCommand="Grid1_Update"

OnDeleteCommand = "Grid1_delete"

DataKeyField="code_value">

<Columns>

<asp:EditCommandColumn

EditText="Edit"

CancelText="Cancel"

UpdateText="OK"

ItemStyle-Wrap="false"

HeaderText="Edit"

HeaderStyle-Wrap="false"/>

<asp:ButtonColumn Text="Delete" CommandName="Delete"

HeaderText="Delete"/>

<asp:BoundColumn HeaderText="Account #" ReadOnly="true"

DataField="code_value"/>

<asp:TemplateColumn HeaderText="Name" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("code_display") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:RequiredFieldValidator runat=server

controltovalidate=edit_Name

errormessage="Name is required.">*

</asp:RequiredFieldValidator>

<asp:TextBox id="edit_name"

Text='<%# Container.DataItem("code_display") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Group" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("category") %>'

runat="server" />

</ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_group" BorderStyle="None"

Readonly="True" Text='<%# Container.DataItem("code_category") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Type" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("type") %>'

runat="server"/>

</ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_type" BorderStyle="None"

Readonly="True" Text='<%# Container.DataItem("type") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Opening" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("opening") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_opening"

Text='<%# Container.DataItem("opening") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Closing" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("closing") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_closing" BorderStyle="None"

Readonly="True" Text='<%# Container.DataItem("closing") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

</Columns>

<HeaderStyle BackColor="DarkRed" ForeColor="White" Font-Bold="true">

</HeaderStyle >

<ItemStyle ForeColor="DarkSlateBlue">

</ltemStyle>

<AlternatingItemStyle BackColor="Beige">

</AlternatingItemStyle>

</asp:DataGrid>

<!---- insert row logic----->

<asp:Panel id="AddPanel" runat="server" Visible="false">

Add a New Account:

Name:

<asp:TextBox id="acode_display" runat="server" />

<asp:RequiredFieldValidator runat=server

controltovalidate=acode_display

errormessage="Name is required.">*

</asp:RequiredFieldValidator>

Group:

<asp:DropDownList DataTextField = "code_display"

```
DataValueField = "code_value" id="acode_category"
```

runat="server" />

```
Opening Value:
```

```
<asp:TextBox id="aopening" value = "0" runat="server" />
```

```
<asp:Button id="SubmitDetailsBtn" text="Submit" onclick="add_Click"
```

```
runat="server" />
```

```
</asp:Panel>
```

```
<!----- Insert Logic ends ----->
```

</form>

</body>

</html>

```
Masters3.vb is the Code Behind file and is as follows:
Masters3.vb (Code Behind)
```

Option Strict Off

Imports System Imports System.Collections Imports System.Text Imports System.Data Imports System.Data.OleDb Imports System.Web.UI Imports System.Web.UI

Public Class BaseClass

Inherits System.Web.UI.Page

Protected Grid1 as DataGrid

Protected Message as label

Protected acode_category as dropdownlist

Protected AddPanel as Panel

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

Connstr = Connstr + " Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

if NOT (isPostBack)

rebind

end if

End Sub

Sub ReBind()

'DataSetCommand

```
SQL = "select m.*, g.code_display as category "
```

SQL = SQL + "from masters m, groups g "

SQL = SQL + " where m.code_category = g.code_value"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "masters")

'Binding a Grid

Grid1.DataSource=ds.Tables("masters").DefaultView

Grid1.DataBind()

SQL = "Select * from groups order by code_display"

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "groups")

'populate drop down list

acode_category.DataSource=ds.Tables("groups").DefaultView

acode_category.DataBind()

hidePanel()

End Sub

Sub Grid1_Edit(Sender As Object, E As DataGridCommandEventArgs)

Grid1.EditItemIndex = E.Item.ItemIndex

ReBind()

End Sub

Sub Grid1_Cancel(Sender As Object, E As DataGridCommandEventArgs)

Grid1.EditItemIndex = -1

ReBind()

End Sub

Sub hidePanel()

If AddPanel.visible = true then

AddPanel.visible = false

end if

End Sub

Sub RunSql(sql as string)

try

Dim mycommand2 As New OleDbCommand(sql,myConnection)

myConnection.Open()

myCommand2.ExecuteNonQuery()

myConnection.Close()

'turn off editing

Grid1.EditItemIndex = -1

Catch ex As OleDbException

' SQL error

Dim errItem As OleDbError

Dim errString As String

For Each errItem In ex.Errors

errString += ex.Message + "
>"

Next

```
Message.Text = "SQL Error.Details follow:<br/><br/>/>" & errString
```

Message.Style("color") = "red"

Catch myException as Exception

Response.Write("Exception: " + myException.ToString())

```
Message.Style("color") = "red"
```

End try

rebind

response.write(sql)

End Sub

End Class

Chapter 16: Transactions

Overview

The Personal Finance Manager maintains cash, bank, and credit card transactions. Each transaction will always have two effects: a debit entry and a credit entry. Debits and credits are applied according to some basic accounting rules. A detailed discussion of

these rules is beyond the scope of this book. However, I shall outline the rules that are relevant to creating transaction entries in the Personal Finance Manager. The Personal Finance Manager deals with two basic transactions—deposits and withdrawals. The rules for these are summarized in <u>Table 16.1</u>.

Table 16.1 Rules for Deposits and W	lithdrawals	
Transaction	Debit	Credit
Deposit	Bank, Cash or Credit Card	Income (exam ple: Salary or Interes t)
Withdrawal	Expense (exampl e: Rent or Utilities)	Bank, Cash or Credit Card

Inserting and Updating Transactions

Like the masters table, I have a procedure that inserts or updates a transaction's entry in the tr_header and the transactions table. This is the procedure p_trans, the listing of which is as follows:

Stored Procedure p_trans

create procedure p_trans

@date datetime,

@ref varchar(30) = NULL,

 $@dr_amount money = 0,$

@cr_amount money =0,

@posted_to integer,

@id char(3),

@doc_no integer = NULL,

@narr varchar(150) = NULL

as

Author: Hersh Bhasin

This procedure creates or modifies a transaction record.

Each transaction record will have a entry in tr_header and two records (a debit and a credit record) in the tranasaction table. Usage: To Insert a record: call with a null doc_no to insert example : exec p_trans @date="01/01/2001", @ref="test", @code_value = 1, @dr_amount = 10, @cr_amount=0, @posted_to = "Sales a/c" ,@id="RPT",@doc_no=Null To modify a record: call with an existing doc_no: example : exec p_trans @date="01/01/2001", @ref="test", @code_value = 1, @dr_amount = 10, @cr_amount=0, @posted_to = "Sales a/c" ,@id="RPT",@doc_no=50

DECLARE @II_doc integer

DECLARE @ret integer

DECLARE @code_value integer

/*

Get the selected cash/bank account:

The user makes a selection from selection.aspx

and tblselection is updated with the code_value

*/

Select @code_value = selection from tblSelection

BEGIN TRANSACTION

IF isnull($@doc_no,0$) = 0

--INSERT—

BEGIN

-SafeGuard : Check if tranaction with same ref# exists. If so do not insert

select @ret = count(*) from tr_header where ref = @ref

```
if @ret > 0
BEGIN
   --raiserror (53000, 1,16)
   GOTO doerror
END
Select @II_doc = isnull(max(doc_no),0)+1 from tr_header
IF @@ERROR != 0
Begin
   GOTO doerror
End
```

END

ELSE

----- UPDATE ------

BEGIN

SELECT @II_doc = @doc_no

Delete from transactions where doc_no = @doc_no

IF @@ERROR != 0

Begin

GOTO doerror

End

Delete from tr_header where doc_no = @doc_no

IF @@ERROR != 0

Begin

GOTO doerror

End

END

BEGIN

INSERT INTO tr_header (id, date,ref, doc_no ,narr) VALUES

```
(@id, isnull(@date,getdate()),@ref, @ll_doc, @narr)
```

```
IF @@ERROR != 0
```

Begin

GOTO doerror

End

INSERT INTO transactions (doc_no, dr_amount, cr_amount,

code_value, sr_no,posted_to)

VALUES

```
( @II_doc, isnull(@dr_amount,0), ISNULL(@cr_amount,0),
```

```
@code_value, 1 ,@posted_to)
```

IF @@ERROR != 0

Begin

GOTO doerror

End

```
INSERT INTO transactions ( doc_no, dr_amount, cr_amount,
```

```
code_value, sr_no, posted_to )
```

VALUES

(@II_doc, ISNULL(@cr_amount,0),ISNULL(@dr_amount,0),

@posted_to, 2 ,@code_value)

IF @@ERROR != 0

Begin

GOTO doerror

End

END

COMMIT TRANSACTION

SELECT 0

GOTO doreturn

doerror:

Rollback TRANSACTION

doreturn:

RETURN 0

SELECT -100

go

This procedure gets called from a DataGrid whenever a new transaction is added or an existing transaction is modified. To *add* (insert) a new record, you will pass an null document number (i.e. $doc_no = null$) to the stored procedure. In this case, the procedure selects the maximum doc_no , increments it by one and stores it in the variable @ll_doc. A tr_header record, having a doc_no equal to @ll_doc, is created with the passed parameters.

To *modify* (update) an existing transaction, you pass the doc_no of the transaction to be modified to the procedure. The procedure stores the passed document number to the variable <code>@ll_doc</code>. It then deletes the transactions with this doc_no because they will be re-created with the passed parameters.

You might wonder why we have to delete and then reinsert the records instead of using an update statement. The reason for this process is that there are triggers associated with this table, which updates the closing balance field in the masters table (I will discuss the triggers in the next section). If you modify the account to which the transaction is debited or credited (say you want to change the transaction account from Rent to Utilities), you will have to reinstate the Rent account closing balance to its state prior to the transaction. You will also have to update the closing balance of the Utilities account to reflect this transaction. This is simple to accomplish if you delete and reinsert the transaction. The delete trigger on the transactions table will reinstate the Rent account to its original value. The insert trigger on the transactions table will update the closing balance figure of the Utilities account with the transaction amount. For both the insert and the update modes, the procedure then creates two equal and opposite transactions in the transactions table. The doc no field for both the transaction records is the number stored in the variable @ll_doc. The two Master accounts affected are specified by the ecode value parameter and the eposted to parameter. The first transaction is given a sn of 1, the second a sn of 2 (if you remember, the primary key of the transactions table is doc no + sn. This procedure in effect creates two records that have the same doc_no but sequential sn's, thus creating two unique records). The dr_amount and the cr_amount in the first transaction are switched and made the cr amount and the dr amount in the second transaction. In this way, this procedure creates an equal debit and credit transaction.

Updating the Closing Balance Field in the Masters Table

I have an insert, update, and delete trigger on the transactions table. Each time a record is added, deleted, or updated, these triggers update the closing balance field of the appropriate account. The advantage of this technique is that we have the closing balances ready at any time and our reports can be compiled quickly. If I did not follow

this technique, each time I have to display a report, I would have to sum the debits and credits in the transactions table. In a large database, where I have to add hundreds of rows, my session would time out, and such an application would be impossible to implement. The triggers are as follows:

insert_mstr
is an insert trigger on the table transactions. Its code is as follows:
insert_mstr

CREATE TRIGGER insert_mstr ON transactions for insert as

Declare @sql varchar(200)

DECLARE @mtype char(1)

DECLARE @amount money

SELECT @mtype = masters.type

FROM masters, inserted

WHERE (masters.code_value = inserted.code_value)

SELECT *

into #temp

from inserted

```
If @mtype = 'A' or @mtype = 'E'
```

BEGIN

```
SELECT @amount = ISNULL(#temp.dr_amount,0) - ISNULL(#temp.cr_amount,0)
```

FROM #temp

END

ELSE

BEGIN

SELECT @amount = ISNULL(#temp.cr_amount,0) - ISNULL(#temp.dr_amount,0)

FROM #temp

END

UPDATE MASTERS

SET closing = closing + @amount

FROM masters, #temp WHERE (masters.code_value = #temp.code_value)

update_mstr is an update trigger on the table transactions. Its code is as follows: update_mstr

CREATE TRIGGER update_mstr ON transactions for update as

Declare @sql varchar(200)

DECLARE @mtype char(1)

DECLARE @amount money

SELECT @mtype = masters.type

FROM masters, inserted

WHERE (masters.code_value = inserted.code_value)

SELECT *

into #temp

from inserted

SELECT *

into #t2

from deleted

If @mtype = 'A' or @mtype = 'E'

BEGIN

SELECT @amount = ISNULL(#temp.dr_amount,0)

- ISNULL(#temp.cr_amount,0)

- ISNULL(#t2.dr_amount,0) + isnull(#t2.cr_amount ,0)

FROM #temp, #t2

WHERE #temp.code_value = #t2.code_value

END

ELSE

BEGIN

SELECT @amount = ISNULL(#temp.cr_amount,0)

- ISNULL(#temp.dr_amount,0)

- ISNULL(#t2.cr_amount,0) + isnull(#t2.dr_amount ,0)

FROM #temp, #t2

WHERE #temp.code_value = #t2.code_value

END

UPDATE Masters

SET Closing = Closing + @amount

FROM masters, #temp WHERE (masters.code_value = #temp.code_value

 $\underline{\texttt{delete_mstr}}$ is a delete trigger on the table <code>transactions</code>. Its code is as follows: $\underline{\texttt{delete_mstr}}$

CREATE TRIGGER delete_mstr ON transactions for delete as

Declare @sql varchar(200)

DECLARE @mtype char(1)

DECLARE @mmonth char(3)

DECLARE @amount money

DECLARE @mstr_amount money

SELECT *

into #temp

from deleted

UPDATE Masters

SET Closing = isnull(Closing,0)-

(ISNULL(t.dr_amount,0)-ISNULL(t.cr_amount,0))

FROM masters m, #temp t

WHERE m.code_value = t.code_value

AND m.type in("A","E")

UPDATE Masters

SET Closing = isnull(Closing,0)-

(ISNULL(t.cr amount,0) - ISNULL(t.dr amount,0))

FROM masters m, #temp t

WHERE m.code_value = t.code_value

AND m.type in("I","L")

Discussion on the Triggers

Microsoft SQL Server maintains an inserted and a deleted table that is used with triggers. An inserted table is a SQL Server table that holds the inserted values in case of an insert statement or the updated values in case of an update statement. A deleted table is a Microsoft SQL Server table that holds the original values in case of an update statement or the deleted value in case of a delete statement. These tables have the same fields as the table it references, which in this case is the transactions table. We can join the inserted or deleted table with any other table. This is the logic followed by the triggers on an insert, update, or delete.

- Each of these triggers looks at the type (that is, "A", "L", "I", "E").
- If the Masters type is a debit type (that is, A or E), the formula for the closing balance is:

Closing balance = dr_amount-cr_amount

If the type is credit, the formula for the closing balance is:

Closing balance = cr amount - dr amount

Table 16.2 summarizes the formula for the closing balance calculations used by these three triggers.

ble 16.2 Closing Balance Calculations				
Action	Trigger	Formula for updating Masters closing balance		
Insert	insert_mstr	<pre>For A, E: closing+ (inserted.dr_amount - inserted.cr_amount) For I, L: closing+ (inserted.cr_amount - inserted.dr_amount)</pre>		
Update	update_mstr	<pre>A, E: closing+(inserted.dr_amo unt - inserted.cr_amount) (deleted.dr_amount - deleted.cr_amount) I,L: closing+(inserted.cr_amo unt -</pre>		

Table 16.2 Closing	able 16.2 Closing Balance Calculations			
Action	Trigger	Formula for updating Masters closing balance		
		inserted.dr_amount)- (deleted.cr_amount - deleted.dr_amount)		
Delete	delete_mstr	<pre>For A, E: closing - (deleted.dr_amount - deleted.cr_amount) For I, L: closing - (deleted.cr_amount - deleted.dr_amount)</pre>		

Transaction Maintenance

Transaction maintenance involves adding, modifying, and deleting transactions. This implementation comprises two web forms, one Code-Behind form, and one stored procedure. These are as follows:

- The Selection Web Form (selection.aspx).
 The Transactions Web Form (transactions.aspx) and the Code-Behind form (transactions.vb).
- 3. Stored Procedure (p_trans).

The Selection Form

The Selection form is a simple form that displays a drop-down list of all the bank, cash, or credit card master accounts defined in the system. The user chooses the appropriate financial account that he wants to work with and clicks on a Submit button. This takes him to the Transactions web form where the actual processing takes place. Figure 16.1 shows what the Selection form looks like.





The code listing of the Selection form is as follows: Selection.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server"> Dim myConnection As OleDbConnection Dim myCommand As OleDbDataAdapter Dim ds As New DataSet Dim ConnStr As String Dim SQL As String Sub Page_Load(Source As Object, E As EventArgs) ConnStr = "Provider=SQLOLEDB; Data Source=(local); " ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;" myConnection = New OleDbConnection(ConnStr) if NOT (isPostBack) rebind end if End Sub Sub SubmitBtn_Click(sender As Object, e As EventArgs) Session("TheSelectionText")= Selection.SelectedItem.Text Session("TheSelectionValue")= Selection.SelectedItem.Value Response.Redirect("Transactions.aspx")

End Sub

Sub ReBind()

'DataSetCommand

SQL = "Select * from masters where code_category in (604,605) "

SQL = SQL + " order by code_display"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "Masters")

Selection.DataSource=ds.Tables("Masters").DefaultView

Selection.DataBind()

End Sub

</script>

<body>

<h3>Financial Account Selection</h3>

<form runat="server" >

Select Cash or Bank Account :<asp:DropDownList

DataTextField = "code_display"

DataValueField = "code_value" id="selection" runat="server" />

<asp:Button text="Go" runat="server" OnClick="SubmitBtn_Click" />

</form>

</body>

</html>

The Selection form presents a drop-down list of all the cash, bank, or credit card accounts defined in the system (that is, master accounts with code_category of 604 or 605). This form posts to the Transactions form, and in the page_load event of this form, the passed code_value is extracted.

The Transaction Form

The Transactions web form is similar to the Masters web form. It enables users to add and modify records. The add functionality is provided by textboxes, residing on a panel which is made visible when the Add button is clicked. A DataGrid implements the "modify" functionality. Figure 16.2 shows what the form looks like. Figure 16.3 shows what it looks like in Add mode.



Figure 16.2: The Transactions form.

Name and Alloca	ni.in	nector, Maste	nt # 2. In Tailbear	a itana					
Ali Colora C Ali Colora C Ali Colora 1 Ali Colora 2 Ali Colora 4	1 2 5	4 Sale 11130-00-50 11130-00-50 111700-00-50 111700-00-50 111700-00-50	Fares Salary Vice Carl Interfuliation	Warts 1 Walk 1 Transfer	Drges (8 42 2	Edd a later. Transaction: Cole (Aspaint): Enformant (Tragami)' nod to angue) Accessit Frankel Tra; Karation: Crawel Answell Extract Answell Extract	Coult in Pland R	

Figure 16.3: The Transactions form in Add mode.

page_load Event

The page_load event creates a new connection to the database. It then extracts the code_value passed to it from the Selection form and stores it into the variable code. This is the primary key of the master's record selected by the user. It then calls the UpdateSelection function with this value.

Transactions.aspx Page_Load Event

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

if NOT (isPostBack)

Dim code as string, display as string

code = Session("TheSelectionValue")

title.text = Session("TheSelectionText")

if code = "" then

response.redirect("selection.aspx")

end if

UpdateSelection(code)

rebind

end if

End Sub

The UpdateSelection Function

The tblSelections table contains a single column called selection. The code_value of the account selected by the user is stored here. This value will later be used in the function ReBind to bind the DataGrid. The following is the script of the function:

Sub UpdateSelection

Sub UpdateSelection(vselection)

sql = "delete from tblSelection "

sql = sql + " insert into tblSelection(selection)"

sql = sql + " values('" + vselection + "')"

runSql(sql)

End Sub

The Rebind Function

The ReBind function binds the DataGrid to a SQL query, first at the Page_Load event, and then whenever the data changes and the grid needs to be refreshed. The following is the script of the function:

Sub ReBind

Sub ReBind()

```
SQL = " select m.code_value,m.code_display,t.*, h.* ,"
```

sql = sql + "(select code_display from masters where code_value = t.posted_to) "

sql = sql + " as posted_display "

sql = sql + " from tr_header h,transactions t, masters m "

sql = sql + " where t.doc_no = h.doc_no "

sql = sql + " and m.code_value = t.code_value"

sql = sql + " and m.code_value = (select selection from tblSelection)"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method to populate dataset

myCommand.Fill(ds, "transactions")

'Binding a Grid

Grid1.DataSource=ds.Tables("transactions").DefaultView

Grid1.DataBind()

populate account selection drop down list which is

visible in the add mode

SQL = "Select * from masters where code_value <> "

SQL = SQL + " (select selection from tblSelection)"

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "masters")

aposted_display.DataSource=ds.Tables("masters").DefaultView

aposted_display.DataBind()

addshow.visible = true

Notice how the code_value is extracted from the tblSelection table in the last line of the query.

sql = sql + "and m.code_value = (select selection from tblSelection)".

The UpdateSelection function had previously inserted the selected code_value in the table tblSelection. This function populates an OleDbDataAdapter with the SQL query, and the Fill method of the OleDbDataAdapter populates the Transactions table of the DataSet with the rows existing in the OleDbDataAdapter. The DataGrid Grid1 is then bound to the default view of the Transactions table of the DataSet. The addshow button is made visible. Clicking on this button, in turn, makes the panel visible and the user can then add a transaction.

The Add Mode

When the addshow button is clicked, the add_show Sub is fired. This Sub simply sets the visible property of the panel AddPanel to visible. This, in turn, makes all the controls residing on this panel visible.

Sub add_show

Sub add_show(Source As Object, E As EventArgs)

AddPanel.visible = true

End Sub

The input controls for the Insert mode have been marked in Transactions.aspx within HTML comment blocks Insert Logic Starts and Insert Logic Ends. Each control has an associated id property, which will be used later to refer to the control.

There is a RequiredFieldValidator attached to the Date and ref columns. The ref needs to be a unique field. The stored procedure p_trans, which gets called when a transaction needs to be added to the database, checks to see whether this field is unique. If not, the procedure will not do anything and will return an error condition. The add_click button is fired when the user clicks on the submitDetails button. This Sub builds a SQL execute query and passes it on to the RunSql function, which in turn executes it. The following is the script for the add_click button: Sub add click

Sub add_click(Source As Object, E As EventArgs)

Dim sql As string

sql = "Execute p_trans @date = '" + adate.text + "',"
sql = sql + "@ref= '" + aref.text + "', @dr_amount = "
sql = sql + adr_amount.text + ", @cr_amount = "
sql = sql + acr_amount.text + ", @posted_to = '"
sql = sql + aposted_display.SelectedItem.value + "',"
sql = sql + "@id = 'RPT', @doc_no = NULL" + ", @narr= '"
sql = sql + anarr.text + "'"
RunSql(sql)
rebind()
hidePanel()

End Sub

The hidePanel function simply hides the panel (by setting its visible property to 0) and sets the values of all the textboxes to "".

The Update Mode

The DataGrid is activated in the Edit mode when the edit link is clicked. The user makes the appropriate changes and clicks on the Ok link. This fires off the Grid1_Update function. The primary key value and other textboxes' values are extracted and a SQL procedure call string is built. This string is passed onto the RunSql function, which makes the actual procedure call.

Sub Grid1_Update

Sub Grid1_Update(sender As Object, e As DataGridCommandEventArgs)

Dim sql As string

Dim vdate As String

Dim ref As String Dim code_value As String Dim dr_amt As String Dim cr_amt As String Dim posted_display As String Dim id As String Dim narr As String Dim myTextBox As TextBox 'This is the key value : Retrieved from the DataKey, since it's a read only field Dim doc_no as string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString myTextBox = E.Item.FindControl("edit_date") vdate = trim(mytextbox.text) myTextBox = E.Item.FindControl("edit_ref") ref = trim(mytextbox.text) myTextBox = E.Item.FindControl("edit_dr_amt") dr_amt = trim(mytextbox.text) myTextBox = E.Item.FindControl("edit_cr_amt") cr amt = trim(mytextbox.text) myTextBox = E.Item.FindControl("edit_posted_display") posted_display= trim(mytextbox.text) myTextBox = E.Item.FindControl("edit_narr") narr = trim(mytextbox.text) 'Now execute stored procedure sql = "Execute p_trans @date = '" + vdate+ "',@ref= '" sql = sql + ref + "', @dr_amount =" sql = sql + dr_amt + ",@cr_amount = " sql = sql + cr_amt +", @posted_to = " + posted_display + "," sql = sql + "@id = 'RPT', @doc_no = " + doc_no + ", @narr= '" + narr+ "'" RunSql(sql)
rebind()

End Sub

Function RunSql

This is a generic function, which executes a SQL Action statement against the database. This is the same function that we discussed in relation to the Masters web form and the code is exactly the same.

The Delete Mode

I have created a ButtonColumn which has a CommandName of "Delete" as follows:

<asp:ButtonColumn Text = "Delete", CommandName = "Delete", HeaderText = "Delete"/>.

In the DataGrid tag, I have specified the <code>OnDeleteCommand</code> to fire the <code>Grid1_delete</code> function. This function gets fired each time the user clicks on the delete hyperlink. The <code>Grid1_delete</code> function sends a SQL delete query to the <code>RunSql</code> function, which deletes all <code>tr_header</code> and <code>transactions</code> records having a document number equal to the clicked <code>doc_no</code>.

The Delete Sub

Sub Grid1_delete(sender As Object, e As DataGridCommandEventArgs)

Dim doc_no as string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString

Dim sql As string

sql = " Delete from transactions where doc_no = " + cstr(doc_no)

sql = sql + " Delete from tr_header where doc_no = " + cstr(doc_no)

RunSql(sql)

rebind()

End Sub

Here is the complete code listing. transactions.aspx

<%@Page Language="VB" Inherits="BaseClass" Src="transactions.vb" %>

```
<%@ Import Namespace="System.Data" %>
```

```
<%@ Import Namespace="System.Data" %>
```

<%@ Import Namespace="System.Data.OleDb" %>

<html>

<script language="VB" runat="server">

Sub Grid1_Update(sender As Object, e As DataGridCommandEventArgs)

Dim sql As string

Dim vdate As String

Dim ref As String

Dim code_value As String

Dim dr_amt As String

Dim cr_amt As String

Dim posted_display As String

Dim id As String

Dim narr As String

Dim myTextBox As TextBox

'This is the key value : Retrieved from the DataKey, since it's a read only field

Dim doc_no as string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString

myTextBox = E.Item.FindControl("edit_date")

vdate = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_ref")

ref = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_dr_amt")

dr_amt = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_cr_amt")

cr_amt = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_posted_display")

posted_display= trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_narr")

narr = trim(mytextbox.text)

'Now execute stored procedure

```
sql = "Execute p_trans @date = '" + vdate+ "',@ref= '"
 sql = sql + ref + "', @dr_amount ="
 sql = sql + dr_amt + ",@cr_amount = "
 sql = sql + cr_amt +", @posted_to = " + posted_display + ","
 sql = sql + "@id = 'RPT', @doc_no = " + doc_no + ", @narr= '" + narr+ "'"
 RunSql(sql)
 rebind()
End Sub
Sub add_click(Source As Object, E As EventArgs)
 Dim sql As string
 sql = "Execute p_trans @date = '" + adate.text + "',"
 sql = sql + "@ref= '" + aref.text + "', @dr_amount = "
 sql = sql + adr_amount.text + ",@cr_amount = "
 sql = sql + acr_amount.text +", @posted_to = "
 sql = sql + aposted_display.SelectedItem.value + "',"
 sql = sql + "@id = 'RPT', @doc_no = NULL" + ", @narr= '"
 sql = sql + anarr.text + ""
 RunSql(sql)
 rebind()
 hidePanel()
End Sub
Sub add_show(Source As Object, E As EventArgs)
 AddPanel.visible = true
End Sub
Sub Grid1_delete(sender As Object, e As DataGridCommandEventArgs)
 Dim doc_no As string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString
 Dim sql As string
 sql = " Delete from transactions where doc_no = " + cstr(doc_no)
 sql = sql + " Delete from tr_header where doc_no = " + cstr(doc_no)
 RunSql(sql)
```

```
rebind()
```

End Sub

</script>

<body style="font: 10pt verdana">

<asp:ValidationSummary runat=server

headertext="There were errors on the page:" />

<form runat="server">

<h3> Transactions for Account #

<asp:Label id="title" runat="server"/>

</h3>

<asp:HyperLink runat="server" Text="Financial Account Selection "

NavigateUrl="selection.aspx"></asp:HyperLink>

<asp:HyperLink runat="server" Text="Masters" NavigateUrl="masters3.aspx">

</asp:HyperLink>

<asp:HyperLink runat="server" Text="Trial Balance" NavigateUrl="TrialBalance.aspx">

</asp:HyperLink>

<asp:HyperLink runat="server" Text="Home" NavigateUrl="default.aspx">

</asp:HyperLink>

<asp:Button id="Addshow" visible = "false" text="New Transaction"

onclick="add_show" runat="server" />

<hr>

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

OnEditCommand="Grid1_Edit"

OnCancelCommand="Grid1_Cancel"

OnUpdateCommand="Grid1_Update"

OnDeleteCommand = "Grid1_delete"

DataKeyField="doc_no">

<Columns>

<asp:EditCommandColumn

EditText="Edit"

CancelText="Cancel"

UpdateText="OK"

ItemStyle-Wrap="false"

HeaderText="Edit"

HeaderStyle-Wrap="false"/>

<asp:ButtonColumn Text="Delete" CommandName="Delete" HeaderText="Delete"/>

<asp:BoundColumn HeaderText="Doc #" ReadOnly="true" DataField="doc_no"/>

<asp:TemplateColumn HeaderText="Ref" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("ref") %>' runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_ref" Text='<%# Container.DataItem("ref") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Date" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("date") %>'runat="server" />

</ltemTemplate>

<EditItemTemplate >

<asp:TextBox id="edit_date" BorderStyle="None" Readonly="True"

Text='<%# Container.DataItem("date") %>' runat="server" />

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Payee" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("posted_display") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_posted_display"

Text='<%# Container.DataItem("posted_to") %>'

runat="server" ReadOnly="true" BorderStyle="None" />

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Narration" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("narr") %>' runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_narr" Text='<%# Container.DataItem("narr") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Deposit" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("dr_amount") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_dr_amt"

Text='<%# Container.DataItem("dr_amount") %>' runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Payment" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("cr_amount") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_cr_amt"

Text='<%# Container.DataItem("cr_amount") %>' runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

</Columns>

<HeaderStyle BackColor="DarkRed" ForeColor="White" Font -Bold="true">

</HeaderStyle>

<ItemStyle ForeColor="DarkSlateBlue">

</ltemStyle>

<AlternatingItemStyle BackColor="Beige">

</AlternatingItemStyle>

</asp:DataGrid>

<! ----insert row logic----->

<asp:Panel id="AddPanel" runat="server" Visible="false">

Add a New Transaction:

Date (Required):

<asp:TextBox id="adate" runat="server" value = "" />

<asp:RequiredFieldValidator runat=server

controltovalidate=adate

errormessage="Date is required.">*

</asp:RequiredFieldValidator>

Reference (Required/ must be unique):

<asp:TextBox id="aref" value = "" runat="server" />

<asp:RequiredFieldValidator runat=server

controltovalidate=aref

errormessage="A unique reference # is required.">*

</asp:RequiredFieldValidator>

Account Posted To:

<asp:DropDownList DataTextField = "code_display"

DataValueField = "code_value" id="aposted_display" runat="server" />

Narration:

<asp:TextBox id="anarr" value = "" runat="server" />

Deposit Amount:

```
<asp:TextBox id="adr_amount" value = 0 runat="server" />
      Payment Amount: 
       <asp:TextBox id="acr_amount" value = 0 runat="server" />
      <asp:Button id="SubmitDetailsBtn" text="Submit" onclick="add_Click"
         runat="server" />
       </asp:Panel>
    <!----> Insert Logic ends ----->
   <hr>
  <asp:Label id="Message" runat="server"/>
 </form>
</body>
</html>
```

Transactions.vb is the Code Behind file. It has the following code: Transactions.vb (Code Behind)

Option Strict Off

Imports System

Imports System.Collections Imports System.Text Imports System.Data Imports System.Data.OleDb Imports System.Web.UI Imports System.Web.UI.WebControls Public Class BaseClass Inherits System.Web.UI.Page Protected Grid1 as DataGrid Protected Message as label, title as label Protected aposted_display as dropdownlist, selection as dropdownlist Protected AddPanel as Panel Protected adate as TextBox, aref as TextBox, adr_amount as TextBox Protected acr_amount as TextBox , anarr as TextBox Protected addshow as button Dim myConnection As OleDbConnection Dim myCommand As OleDbDataAdapter Dim ds As New DataSet Dim ConnStr As String **Dim SQL As String** Sub Page_Load(Source As Object, E As EventArgs) ConnStr = "Provider=SQLOLEDB; Data Source=(local); ConnStr = ConnStr + " Initial Catalog=ASPNET;User ID=sa;" " myConnection = New OleDbConnection(ConnStr) if NOT (isPostBack) Dim code as string, display as string code = Request.Form("Selection") title.text = code if code = "" then response.redirect("selection.aspx")

end if

UpdateSelection(code)

rebind

end if

End Sub

Sub ReBind()

- SQL = " select m.code_value,m.code_display,t.*, h.* ,"
- sql = sql + "(select code_display from masters where code_value = t.posted_to) "
- sql = sql + " as posted_display "
- sql = sql + " from tr_header h,transactions t, masters m "
- sql = sql + " where t.doc_no = h.doc_no "

sql = sql + " and m.code_value = t.code_value"

sql = sql + " and m.code_value = (select selection from tblSelection)"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method to populate dataset

myCommand.Fill(ds, "transactions")

'Binding a Grid

Grid1.DataSource=ds.Tables("transactions").DefaultView

Grid1.DataBind()

'populate account selection drop down list

' which is visible in the add mode

SQL = "Select * from masters where code_value <> "

SQL = SQL + " (select selection from tblSelection)"

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "masters")

aposted_display.DataSource=ds.Tables("masters").DefaultView

aposted_display.DataBind()

addshow.visible = true

End Sub

Sub Grid1_Edit(Sender As Object, E As DataGridCommandEventArgs)

Grid1.EditItemIndex = E.Item.ItemIndex ReBind() End Sub Sub Grid1_Cancel(Sender As Object, E As DataGridCommandEventArgs) Grid1.EditItemIndex = -1 ReBind() End Sub Sub RunSql(vsql as string) try 'sql = "Execute p_test " + vkey Dim mycommand2 As New OleDbCommand(vsql,myConnection) myConnection.Open() myCommand2.ExecuteNonQuery() myConnection.Close() 'turn off editing Grid1.EditItemIndex = -1 Catch ex As OleDbException ' SQL error Dim errItem As OleDbError Dim errString As String For Each errItem In ex.Errors errString += ex.Message + "
>" Next Message.Style("color") = "red" Response.write("DataBase Error :" + errString) Catch myException as Exception Response.Write("Exception: " + myException.ToString()) Message.Style("color") = "red" Finally message.text = vsql

```
end try
```

End Sub

Sub hidePanel()

if AddPanel.visible = true then

AddPanel.visible = false

'reset values

adate.text = ""

aref.text = ""

adr_amount.text = ""

acr_amount.text = ""

anarr.text = ""

end if

End Sub

Sub UpdateSelection(vselection)

sql = "delete from tblSelection "

sql = sql + " insert into tblSelection(selection)"

sql = sql + " values('" + vselection + "')"

runSql(sql)

End Sub

End Class

Chapter 17: The Trial Balance Report

Overview

The Trial Balance is the main accounting report and is the basis for all other financial reports like the balance sheet, the profit and loss, and the cash flow statement. This report presents the closing balance of all master accounts in a tabular layout with the debit accounts on one side and the credit accounts on the other. The Acid test to show that all our transaction entries are correct is ensuring that the Trial Balance's debits and credits match.

Figure 17.1 shows what the Trial Balance web form looks like.

nis (2) Casconice Linis (e]ree Honal @]teach	thane Subresson Stes @]Windows Media @]Windows	1.44
Trial Balance			2
Masters Transactions	Hone		
Account	Debit Amount	Fredit Amount	
Carb in Hand	2	3	
Kells Farge Checking	40	3	
visa Card	40	2	
laiary	6	100	
interest Income	£	3	
kent	30	3	
Rities	6	3	
Redical Expenses		3	
Intertainment	2	3	
otal	100	100	

Figure 17.1: The Trial Balance.

The Trial Balance web form is comprised of two DataGrids. The first gets the detail rows of the Trial Balance and is bound to the following query:

SELECT code_display, closing,

```
dr_amount = CASE type
WHEN 'A' THEN closing
WHEN 'E' THEN closing
ELSE 0
END,
cr_amount = CASE type
WHEN 'I' THEN closing
WHEN 'L' THEN closing
ELSE 0
END
FROM Masters
```

I need to show debits and credits in two separate columns; hence, using a case statement I separate the debits and credits. Remember that assets and expenses are of type debit and liabilities and income are of type credit. The second grid displays the grand total of all debits and credits. This is bound to the following SQL query:

SELECT 'Total' as nothing,

(Select sum(closing) From masters where type in('A','E')) as dr_total ,

(Select sum(closing) From masters where type in('I','L')) as cr_total

Here is the complete code listing for the TrialBalance.aspx web form. TrialBalance.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<html>

```
<script language="VB" runat="server">
```

Dim myConnection As OleDbConnection

Dim myCommand As OleDbDataAdapter

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

myConnection = New OleDbConnection(ConnStr)

if NOT (isPostBack)

rebind

end if

End Sub

Sub ReBind()

'DataSetCommand

sql = "SELECT code_display, closing, "

sql = sql + " dr_amount = CASE type WHEN 'A' THEN "

sql = sql + " closing WHEN 'E' THEN closing ELSE 0 END, "

sql = sql + " cr_amount = CASE type WHEN 'l' "

sql = sql + " THEN closing WHEN 'L' THEN closing ELSE 0 END "

sql = sql + " From Masters"

myCommand = New OleDbDataAdapter(SQL, myConnection)

'use Fill method of DataSetCommand to populate dataset

myCommand.Fill(ds, "Masters")

'Binding a Grid

Grid1.DataSource=ds.Tables("Masters").DefaultView

Grid1.DataBind()

'totals

sql = "SELECT 'Total' as nothing ,"

sql = sql + " (Select sum(closing) From masters "

sql = sql + " where type in('A','E')) as dr_total , "

sql = sql + " (Select sum(closing) From masters "

sql = sql + " where type in('l','L')) as cr_total "

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "totals")

'Binding a Grid

Grid2.DataSource=ds.Tables("totals").DefaultView

Grid2.DataBind()

End Sub

</script>

<body>

<h3>Trial Balance </h3>

<form runat=server>

<asp:HyperLink runat="server" Text="Masters"

NavigateUrl="masters3.aspx"></asp:HyperLink>

<asp:HyperLink runat="server" Text="Transactions"

NavigateUrl="selection.aspx"></asp:HyperLink>

<asp:HyperLink runat="server" Text="Home"

NavigateUrl="default.aspx"></asp:HyperLink>

>

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt">

<Columns>

<asp:BoundColumn HeaderText="Account" DataField="code_display">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Debit Amount" DataField="dr_amount" >

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Credit Amount" DataField="cr_amount" >

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

</Columns>

<HeaderStyle BackColor="teal" ForeColor="white" Font-Bold="true">

</HeaderStyle>

<ItemStyle ForeColor="DarkSlateBlue">

</ltemStyle>

<AlternatingItemStyle BackColor="Beige">

</AlternatingItemStyle>

</asp:DataGrid>

<!---->

<asp:DataGrid id="Grid2" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt">

<Columns>

<asp:BoundColumn HeaderText="" DataField="nothing">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

</Columns>

<Columns>

<asp:BoundColumn HeaderText="" DataField="dr_total">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="" DataField="cr_total" >

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

</Columns>

<ItemStyle BackColor="teal" foreColor="white" Font-Bold="true">

</ltemStyle>

</asp:DataGrid>

</form>

</body>

</html>

Project 1 Summary

In this Project, I took a traditional client/server application and revamped it for the Web. Applications that work off the Web offer significant advantages over their client/server brethren. This accounting application is totally wireless. You don't need to lay out network cables to connect it to the database. You will notice that I have relegated most of the script processing to stored procedures and triggers. This has enabled me to have a very thin web form.

In <u>Project 2</u> of this book, I will incorporate web services in this application. I can then harness the full power of the Web as well as that of a client/server product.

Project 2: Web Services

Chapter List

- <u>Chapter 18:</u> Creating a Generic Database Web Service
- <u>Chapter 19</u>: Designing a Navigation System
- <u>Chapter 20:</u> Incorporating Web Services in the Chart of Accounts Form
- <u>Chapter 21:</u> Incorporating Web Services in the Transactions Form
- <u>Chapter 22:</u> Incorporating Web Services in the Trial Balance

Project 2 Overview

The Personal Finance Manager described in the preceding project showed how we can Web-enable an accounting application and make it accessible through the Internet. The consequences of this approach are far-reaching. Consider the case of a car manufacturer who has a large number of ancillary companies manufacturing components, such as air filters, wind screens, tires, and so on. These ancillary companies are spread all over the world. This car manufacturer uses "Just in Time" manufacturing techniques and the ancillary companies only produce (and ship) as much as can be consumed by the manufacturer in a given period. This removes the necessity of holding inventory both by the supplier and the car manufacturer. Say an overseas company manufactures air filters for the car manufacturer and will manufacture them accordingly. To do this, the overseas company needs access to the daily production requirements of the car manufacturer, who, in turn, requires access to the production and inventory records of the supplier. Do we need to set up a satellite link between the companies to enable their databases to communicate?

A retail chain has store locations all over the U.S. Each time a customer makes a purchase at the store, the item sold must be removed from inventory and the dollar amount of the purchase incorporated in the financial books as a sale. Suppose all the constituent stores of the chain read and write to one central database. Do we set up network infrastructure between all the stores for the communication to take place? And, at what cost?

In both the preceding scenarios, we need to set up lines of communication between databases located in distant locations. Traditionally, we would have to set up network communication infrastructure to make this happen. However, using web services, we can link them using the Internet.

The basic requirement of an application that interacts with a database is to select, insert, update, and delete records from the database. Using web services, we can encapsulate the database interaction logic as a series of methods that are called over the Internet. When we have the ability to interact with the database over the Web, we don't require expensive satellite or "wire" links between applications and databases. The Personal Finance Manager developed in the previous project was tightly coupled to the ASP.NET technology and worked in the browser. Tools such as Visual Basic.NET and C# enable us to build applications that are user-friendly and have a rich user interface. For example, users have grown accustomed to seeing toolbars, MDI (multiple document interface), and other graphical niceties, which a browser is not able to provide. Won't it be nice to build applications using one of these tools and still be able to communicate over the Internet? We would then be able to build applications with a rich graphical user interface and not have to worry about setting up elaborate network systems for communication. Web services enables us to do just that. As I will show you in this project, we can build generic database access services which can then be used with an application built in Visual Basic.NET, C#, or ASP.NET. I will show you how to use this web service with the Personal Finance Manager using ASP.NET. You can develop a feature-rich application (with all the associated graphical niceties) in Visual Basic.NET or C# and still use this web service.

Chapter 18: Creating a Generic Database Web Service

Overview

In <u>Chapter 8</u>, I showed you how to create a generic database business object for communicating with a database. This object had useful functions for executing action

queries, such as insert, update, and delete as well as a generic routine for returning a DataSet, based on any user-supplied SQL query. This DataSet then could be used to bind an ASP.NET server control. I will now show you how to convert this business object into a web service. Various accounting modules can then use the functionality provided by the service. In Figure 18.1, I show how the two projects developed in this book (the Personal Finance Manager module and the Inventory module, which will be developed in Part IV) interact with the web service. The supporting example files for this example are located in the folder samples\SqlService of this chapter on the book's Web site at www.premierpressbooks.com/downloads.asp.



Figure 18.1: Interacting with the web service.

I will now walk you through the process of creating the web service. This comprises a number of steps, as follows:

1. Create the web service asmx file: Use NotePad to create a file called SQLService.asmx, which has the following script:

```
SQLService.asmx
< @ WebService Language="VB" Class="SQLService"%>
Imports System
Imports System.Web.Services
Imports System.Data
Imports System.Data.OleDb
Imports System.Text
Public Class SQLService: Inherits WebService
 <WebMethod()> Public Function TestFunction (vInput as Boolean) As
String
  If (vInput = TRUE) Then
   TestFunction = "It is the truth ... "
  Flse
   TestFunction = "False!False!False"
  End if
 End Function
 <WebMethod()> Public Function add( a as integer, b as integer) as string
  add = cstr(a+b)
 End function
 <WebMethod()> Public Function Populate(ConnStr as string, SQL as string)
As DataSet
  Dim dv As DataView
  Dim i As integer
```

```
Dim myConnection As OleDbConnection
Dim myCommand As OleDbDataAdapter
Dim ds As New DataSet
myConnection = New OleDbConnection(ConnStr)
myCommand = New OleDbDataAdapter(SQL, myConnection)
myCommand.Fill(ds, "vTable")
'Populate = ds.Tables("vTable").DefaultView
Populate = ds
End Function
```

<WebMethod()>PUBLIC Function RunSql (ConnStr as string, vsql as string) as String Dim Message As string try message = "Success" Dim myConnection As OleDbConnection myConnection = New OleDbConnection(ConnStr) Dim mycommand As New OleDbCommand(vsql,myConnection) myConnection.Open() myCommand.ExecuteNonQuery() myConnection.Close() Catch ex As OleDbException Dim errItem As OleDbError Dim errString As String For Each errItem In ex.Errors errString += ex.Message + " " Next Message = "SQL Error.Details follow:

br/>" & errString Catch myException as Exception message = "Exception: " + myException.ToString() End try RunSql = message End Function End Class

This is exactly the same code that I developed in <u>Chapter 8</u>, hence I am not discussing it in detail here. The differences are that this file has an .asmx extension, the class inherits from WebService, a WebService attribute is added to the form header, the System.Web.Services namespace is imported and each function is prefixed with a <WebMethod() > tag. The WebMethod tag indicates to the ASP.NET run time that the method in question can be called over the Web.

- Create the WSDL file: Open SQLService.asmx so that it goes through IIS (such as http://localhost/your virtual directory/SQLService.asmx). Click on Show WSDL. Save the resultant file as SQLService.wsdl. Note that you can do the same thing by browsing to http://localhost/your virtual directory/SQLService.asmx?wsdl.
- Create and compile the proxy: Run msqlProxy.bat, which will create the proxy SQLService.vb in the local folder and compile the DLL SQLService.dll to the bin folder.

msqlProxy.bat

REM ------Make Proxy------

wsdl.exe /I:VB /n:NameSpaceHersh /out:SqlService.vb SqlService.wsdl

REM ------Compile Proxy------

Rem Remember to change outdir variable to point to your bin folder

set outdir=g:\AspNetSamples\bin\SQLService.dll

set

assemblies=System.dll,System.Web.dll,System.Data.dll,System.Web.Service s.dll,System.Xml.dll

vbc /t:library /out:%outdir% /r:%assemblies% SQLService.vb

pause

4. Test the service: The form that I have written to test the service is SQLService.aspx, which has the following code:

```
SQLService.aspx
```

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>
<%@ Import Namespace="NameSpaceHersh" %>
<html>
 <script language="VB" runat="server">
  Dim SQL as string, vcn as string
  Sub Page_Load(Source As Object, E As EventArgs)
   vcn= "Provider=SQLOLEDB; Data Source=(local); Initial
Catalog=ASPNET;User ID=sa;"
   if NOT (isPostBack)
     rebind
   end if
   End Sub
  Sub Show_Click(Sender As Object, E As EventArgs)
   Message.Text = "Masters Table Displayed... "
   ReBind
  End Sub
  Sub Insert_click(Sender As Object, E As EventArgs)
   sql = "Insert into Masters(code_display,code_category,type)"
   sql = sql + "Values ('test',701,'E')"
   RunSQL(sql)
   rebind
   Message.Text = "..Inserted test record... "
  End Sub
  Sub Delete_click(Sender As Object, E As EventArgs)
   sql = "delete from masters where code_display = 'test'"
   RunSQL(sql)
   rebind
   Message.Text = "...Deleted all test records..."
  End Sub
  Sub Update_Click(Sender As Object, E As EventArgs)
```

sql = "UPDATE Masters Set Opening = 90 WHERE code_display = 'test'" RunSQL(sql) rebind Message.Text = "...Updated all test records: Set closing balance = 90 ...! " End Sub Sub ReBind() Dim t As NameSpaceHersh.SqlService = New NameSpaceHersh.SqlService() Dim vsql as string Dim ds as DataSet vSQL = "select * from Masters" ds = t.Populate(vcn, vSql) DataGrid1.DataSource=ds.Tables("vTable").DefaultView DataGrid1.DataBind() End Sub Function RunSQL (vSQL as String) Dim t As NameSpaceHersh.SqlService = New NameSpaceHersh.SqlService() t.RunSQL(vcn,vSQL) End Function </script> <body> <body style="font: 10pt verdana; background-color:beige"> <form runat=server> <h4> ASP .NET Web Services </h4> <asp:button text="Refresh" Onclick="Show_Click" runat=server/> <asp:button text="Insert" Onclick="Insert_Click" runat=server/> <asp:button text="Update" Onclick="Update_Click" runat=server/> <asp:button text="Delete" Onclick="Delete_Click" runat=server/> <asp:label id="Message" runat=server/> <asp:DataGrid id="DataGrid1" runat="server" /> </form> </body> </html>

In the Page_load event, I specify the connection string as:

vcn= "Provider=SQLOLEDB; Data Source=(local); _

Initial Catalog=ASPNET;User ID=sa;"

The Sub ReBind calls the Populate function of the web service and passes it the query "select * from Masters" as well as the connection string. The Populate function returns a DataSet, the default view of which is bound to a DataGrid as follows:

Sub ReBind()

Dim t As NameSpaceHersh.SqlService = New NameSpaceHersh.SqlService()

Dim vsql as string

```
Dim ds as DataSet
```

vSQL = "select * from Masters"

ds = t.Populate(vcn, vSql)

DataGrid1.DataSource=ds.Tables("vTable").DefaultView

DataGrid1.DataBind()

End Sub

Note that this is a pretty nifty way of calling queries that return data. You can pass any SQL query (the query may be a join on multiple tables) and get a DataSet back, which can then be manipulated in any way. I can also connect to any database as I am passing the database connection string to the function.

I have three buttons and associated click events which pass an insert, an update, and a delete statement respectively to the RunSQL function on the form, which in turn calls the RunSQL function of the web service. Here are the three functions:

Sub Insert_click(Sender As Object, E As EventArgs)

```
sql = "Insert into Masters(code_display,code_category,type)"
```

```
sql = sql + "Values ('test',701,'E')"
```

RunSQL(sql)

rebind

Message.Text = "..Inserted test record... "

End Sub

Sub Delete_click(Sender As Object, E As EventArgs)

```
sql = "delete from masters where code_display = 'test'"
```

RunSQL(sql)

rebind

Message.Text = "...Deleted all test records..."

End Sub

Sub Update_Click(Sender As Object, E As EventArgs)

```
sql = "UPDATE Masters Set Opening = 90 WHERE code_display = 'test'"
```

RunSQL(sql)

rebind

Message.Text = "...Updated all test records: Set closing balance = 90...!

End Sub

The local RunSQL function calls the RunSQL function of the web service and passes it the SQL string and the connection string. The web service function then executes the action query.

Function RunSQL (vSQL as String)

Dim t As NameSpaceHersh.SqlService = New NameSpaceHersh.SqlService()

t.RunSQL(vcn,vSQL)

End Function

Web services is an important element of ASP.NET. The techniques developed in this chapter show how the process of database interaction can be abstracted and encapsulated as a web service. In the subsequent chapters of this project (Chapters 19 through 22), I will explain how the techniques developed in this chapter can be applied to an accounting application.

Chapter 19: Designing a Navigation System

Overview

In <u>Chapter 6</u>, you designed an XML-based site navigation system. You developed a user control which when placed on a Web page generated the navigation links. <u>Figure 19.1</u> shows what the navigation links look like.



Figure 19.1: Navigation links.

The navigation links were stored in an XML file. I will briefly review the user control here because I am going to use it on every form of the project.

Navigation Links

The navigation links of this project are stored in the file nav.xml.

Nav.xml

<Siteinfo>

<site>

<sitename>Home</sitename>

```
<siteurl>default.aspx</siteurl>
```

</site>

<site>

<sitename>Masters</sitename>

<siteurl>masters3.aspx</siteurl>

</site>

<site>

<sitename>Transactions</sitename>

<siteurl>selection.aspx</siteurl>

</site>

<site>

<sitename>Trial Balance</sitename>

<siteurl>Trialbalance.aspx</siteurl>

</site>

</Siteinfo>

Each link to be displayed is enclosed within the site node, which has two elements: the site name and the site URL. My user control will display each of these URLs at the top of each page.

The User Control

The user control is described in <u>Chapter 6</u>. To reiterate, you can assign it the GridLines, BorderColor, and CellPadding properties. It reads the XML file and binds a DataList to it. The DataList displays the links that you see on each page. Here is the code:

nav.ascx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.IO" %>

<%@ Import Namespace="System.Drawing" %>

<script language="VB" runat="server">

'Public Variable for each exposed Property

PUBLIC vGridLines As GridLines

PUBLIC vBorderColor as String

PUBLIC vCellPadding As Integer

Sub Page_Load(Source As Object, E As EventArgs) Dim ds As New DataSet Dim fs As filestream Dim xmLStream As StreamReader fs = New filestream(Server.MapPath("nav.xml"), FileMode.Open, FileAccess.Read) xmlStream = new StreamReader(fs) ds.ReadXML(XmlStream) fs.Close() dlist.DataSource=ds.Tables("site").DefaultView dlist.DataBind() dlist.GridLines = vGridLines dlist.BorderColor=System.Drawing.Color.FromName(vBorderColor) dlist.CellPadding=vCellPadding End Sub </script> <asp:DataList runat=server id="dlist" RepeatDirection="horizontal" RepeatMode="Table" Width="100%" BorderWidth="1" Font-Name="Verdana" Font-Size="8pt" HeaderStyle-BackColor="#aaaadd" SelectedItemStyle-BackColor="yellow" ItemStyle-BackColor="antiquewhite" AlternatingItemStyle-BackColor="tan" > <ItemTemplate>

<asp:HyperLink runat="server"

Text='<%# Container.DataItem("sitename") %>'

NavigateUrl= '<%# Container.DataItem("siteurl") %>' />

</ltemTemplate>

</asp:DataList>

Using the Control

Each Web page must register the control by using the following declaration at the top of each page:

<%@ Register TagPrefix="Hersh" TagName="nav" Src="nav.ascx" %>

Within the page, the control is invoked as follows:

<!----- Navigation Start-----> <Hersh:nav id="menu" runat = server vGridlines = Both vBorderColor = "Black"

vCellPadding = 7 />

<!----- Navigation Ends----->

Chapter 20: Incorporating Web Services in the Chart of Accounts Form

Overview

The Personal Finance Manager built in <u>Project 1</u> of this book used the Masters.aspx web form (together with its Code Behind form, Masters.vb) to interact with the masters database table. This web form was developed in <u>chapter 15</u> (Chart of Accounts). In this chapter, I will modify this web form so that it can use the SQLService web service. I will need to change two methods residing in the Code Behind form Masters.vb in order to use the web service. These methods are the ReBind and the RunSql methods, both of which interact with the database.

The ReBind() method was used to bind a DataGrid and a DropDownList control to a database table. The following is the code snippet of the ReBind function, which I am going to change:

The Original ReBind Method

Sub ReBind()

SQL = "select m.*, g.code_display as category "

SQL = SQL + "from masters m, groups g "

SQL = SQL + " where m.code_category = g.code_value"

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "masters")

'Binding a Grid

Grid1.DataSource=ds.Tables("masters").DefaultView

Grid1.DataBind()

SQL = "Select * from groups order by code_display"

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "groups")

'populate drop down list

acode_category.DataSource=ds.Tables("groups").DefaultView

acode_category.DataBind()

hidePanel()

End Sub

To use the web service, I have changed this to what is shown in the following code snippet:

The Mmodified ReBind Method that Uses the Web Services

Sub ReBind()

Dim t As New NameSpaceHersh.SQLService

Dim ds As DataSet

' Bind Grid

SQL = "select m.*, g.code_display as category "

SQL = SQL + "from masters m, groups g "

SQL = SQL + " where m.code_category = g.code_value"

ds = t.Populate(ConnStr, SQL)

Grid1.DataSource=ds.Tables("vTable").DefaultView

Grid1.DataBind()

'populate drop down list

SQL = "Select * from groups order by code_display"

ds = t.Populate(ConnStr, SQL)

acode_category.DataSource=ds.Tables("vTable").DefaultView

acode_category.DataBind()

hidePanel()

End Sub

In the modified ReBind method, I call the web service method called Populate twice, each time passing it a connection string and a SQL query. A DataSet containing the result set is returned from the function, which I use to bind the DataGrid and DropDownList control respectively.

The second function that I need to modify is the RunSql function. This function is used to add, delete, or update a row. The appropriate event handlers pass to this function a SQL action query or a stored procedure name with the appropriate parameters. The original code snippet is listed below:

```
The Original RunSql Method
```

Sub RunSql(sql as string)

try

Dim mycommand2 As New OleDbCommand(sql,myConnection)

myConnection.Open()

myCommand2.ExecuteNonQuery()

myConnection.Close()

'turn off editing

Grid1.EditItemIndex = -1

Catch ex As OleDbException

' SQL error

Dim errItem As OleDbError

Dim errString As String

For Each errItem In ex.Errors

errString += ex.Message + "
>"

Next

Message.Text = "SQL Error.Details follow:

br/>" & errString

Message.Style("color") = "red"

Catch myException as Exception

Response.Write("Exception: " + myException.ToString())

Message.Style("color") = "red"

End try

rebind

response.write(sql)

End Sub

The following code extract shows the modified version of this function: Modified Version of the RunSql Method

Sub RunSql(vSQL as string)

Dim t As New NameSpaceHersh.SQLService

Dim s As string

```
s = t.RunSQL(ConnStr,vSQL)
```

Grid1.EditItemIndex = -1

Rebind

```
if s <> "Success" then
```

```
Message.Text = s
```

Message.Style("color") = "red"

End if

End Sub

This function calls the RunSql function of the web service and passes it the connection string as well as the SQL action query/procedure call as parameters. If the procedure/query was executed successfully, the string Success is returned from the function. Otherwise, the appropriate error string is returned, which is displayed in the browser.

I include the complete listing of the Code Behind file, Masters.vb, after modifying the two methods as discussed above. There is no change made to the web form Masters.aspx. Masters3.vb

Option Strict Off

Imports System

- Imports System.Collections
- Imports System.Text

Imports System.Data

Imports System.Data.OleDb

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class BaseClass

Inherits System.Web.UI.Page

Protected Grid1 as DataGrid

Protected Message as label

Protected acode_category as dropdownlist

Protected AddPanel as Panel

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

if NOT (isPostBack)

rebind

end if

End Sub

Sub ReBind()

Dim t As New NameSpaceHersh.SQLService

Dim ds As DataSet

' Bind Grid

SQL = "select m.*, g.code_display as category "

SQL = SQL + "from masters m, groups g "

SQL = SQL + " where m.code_category = g.code_value"

ds = t.Populate(ConnStr, SQL) Grid1.DataSource=ds.Tables("vTable").DefaultView Grid1.DataBind() 'populate drop down list SQL = "Select * from groups order by code_display" ds = t.Populate(ConnStr, SQL) acode_category.DataSource=ds.Tables("vTable").DefaultView acode_category.DataBind() hidePanel() End Sub Sub Grid1_Edit(Sender As Object, E As DataGridCommandEventArgs) Grid1.EditItemIndex = E.Item.ItemIndex ReBind() End Sub Sub Grid1_Cancel(Sender As Object, E As DataGridCommandEventArgs) Grid1.EditItemIndex = -1 ReBind() End Sub Sub hidePanel() if AddPanel.visible = true then AddPanel.visible = false end if End Sub Sub RunSql(vSQL as string) Dim t As New NameSpaceHersh.SQLService Dim s As string

s = t.RunSQL(ConnStr,vSQL)

Grid1.EditItemIndex = -1

Rebind

if s <> "Success" then

```
Message.Text = s
```

```
Message.Style("color") = "red"
```

End if

End Sub

End Class

Chapter 21: Incorporating Web Services in the Transactions Form

Overview

In <u>Chapter 16</u>, I built the Transactions web form (Transactions.aspx together with the Code Behind form Transactions.vb) for the Personal Finance Manager. In this chapter, I will modify this form so that it uses the SQLService web service for its database interaction.

I will have to modify the ReBind and the RunSql methods, both of which reside in the Code Behind file called Transactions.vb. The aspx form Transactions.aspx remains unchanged.

The ReBind method binds a DataGrid and a DropDownList control to a database table. Here is the original method that I will change.

The Original ReBind Method

Sub ReBind()

sql = " select m.code_value,m.code_display,t.*, h.* ,"

sql = sql + "(select code_display from masters where code_value = t.posted_to) "

sql = sql + " as posted_display "

sql = sql + " from tr_header h,transactions t, masters m "

sql = sql + " where t.doc_no = h.doc_no "

sql = sql + " and m.code_value = t.code_value"

sql = sql + " and m.code_value = (select selection from tblSelection)"

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "transactions")

'Binding a Grid

Grid1.DataSource=ds.Tables("transactions").DefaultView

Grid1.DataBind()

populate account selection drop down list

' which is visible in the add mode

sql = "Select * from masters where code_value <> "

sql = sql + " (select selection from tblSelection)"

myCommand = New OleDbDataAdapter(SQL, myConnection)

myCommand.Fill(ds, "masters")

aposted_display.DataSource=ds.Tables("masters").DefaultView

aposted_display.DataBind()

addshow.visible = true

End Sub

To use the SQLService web service, I have changed this to the following: **The Modified ReBind Method**

Sub ReBind()

Dim t As New NameSpaceHersh.SQLService

Dim ds As DataSet

SQL = " select m.code_value,m.code_display,t.*, h.* ,"

sql = sql + "(select code_display from masters where "

sql = sql + " code_value = t.posted_to) as posted_display "

- sql = sql + " from tr_header h,transactions t, masters m "
- sql = sql + " where t.doc_no = h.doc_no "
- sql = sql + " and m.code_value = t.code_value"
- sql = sql + " and m.code_value = (select selection from tblSelection)"
- ds = t.Populate(ConnStr, SQL)
- Grid1.DataSource=ds.Tables("vTable").DefaultView

Grid1.DataBind()

'populate account(add mode) selection drop down list

'SQL = "Select * from masters"

SQL = "Select * from masters where code_value <> "

SQL = SQL + " (select selection from tblSelection)"

ds = t.Populate(ConnStr, SQL)

aposted_display.DataSource=ds.Tables("vTable").DefaultView

aposted_display.DataBind()

addshow.visible = true

End Sub

In this function, I call the web service method Populate twice, each time passing to it the connection string and a SQL query as parameters. A DataSet containing the database rows is returned from the function, which I use to bind the DataGrid and DropDownList control respectively.

The second function that I need to modify is the RunSql function. This function is used to add, delete, or update a row. The appropriate event handlers pass to this function a SQL action query or a stored procedure name with appropriate parameters. This is the original code snippet:

The Original RunSql Function

Sub RunSql(vsql as string)

try

Dim mycommand2 As New OleDbCommand(vsql,myConnection)

myConnection.Open()

myCommand2.ExecuteNonQuery()

myConnection.Close()

'turn off editing

Grid1.EditItemIndex = -1

Catch ex As OleDbException

' SQL error

Dim errItem As OleDbError

Dim errString As String

For Each errItem In ex.Errors

errString += ex.Message + "
>"

Next

Message.Style("color") = "red"
```
Response.write("DataBase Error :" + errString)
  Catch myException as Exception
    Response.Write("Exception: " + myException.ToString())
     Message.Style("color") = "red"
     Finally
     message.text = vsql
 end try
End Sub
Sub hidePanel()
 if AddPanel.visible = true then
  AddPanel.visible = false
  'reset values
  adate.text = ""
   aref.text = ""
    adr_amount.text = ""
   acr_amount.text = ""
  anarr.text = ""
 end if
```

Here is the modified version of this function: Modified Version of the RunSql Function

Sub RunSql(vsql as string)

Dim t As New NameSpaceHersh.SQLService

Dim s As string

s = t.RunSQL(ConnStr,vSQL)

Grid1.EditItemIndex = -1

Rebind

End Sub

```
if s <> "Success" then
```

Message.Text = s

Message.Style("color") = "red"

End if

End Sub

This function calls the RunSql function of the web service and passes it a connection string as well as a SQL action query/procedure call string as input parameters. If the procedure/query was executed successfully, the string Success is returned from the function. Otherwise, the appropriate error string is returned, which is displayed in the browser.

Here is the listing of the modified Code Behind form Transactions.vb:

Transactions.vb

Option Strict Off

Imports System

Imports System.Collections

Imports System.Text

Imports System.Data

Imports System.Data.OleDb

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class BaseClass

Inherits System.Web.UI.Page

Protected Grid1 as DataGrid

Protected Message as label, title as label

Protected aposted_display as dropdownlist, selection as dropdownlist

Protected AddPanel as Panel

Protected adate as TextBox, aref as TextBox, adr_amount as TextBox

Protected acr_amount as TextBox , anarr as TextBox

Protected addshow as button

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

if NOT (isPostBack)

Dim code As string, display As string

code = Session("TheSelectionValue")

title.text = Session("TheSelectionText")

if code = "" then

response.redirect("selection.aspx")

end if

UpdateSelection(code)

rebind

end if

End Sub

Sub ReBind()

Dim t As New NameSpaceHersh.SQLService

Dim ds As DataSet

SQL = " select m.code_value,m.code_display,t.*, h.* ,"

sql = sql + "(select code_display from masters where "

sql = sql + " code_value = t.posted_to) as posted_display"

sql = sql + " from tr_header h,transactions t, masters m "

sql = sql + " where t.doc_no = h.doc_no "

sql = sql + " and m.code_value = t.code_value"

sql = sql + " and m.code_value = (select selection from tblSelection)"

ds = t.Populate(ConnStr, SQL)

Grid1.DataSource=ds.Tables("vTable").DefaultView

Grid1.DataBind()

```
'populate account(add mode) selection drop down list
 sql = "Select * from masters where code_value <> "
 sql = sql + " (select selection from tblSelection)"
 ds = t.Populate(ConnStr, SQL)
 aposted_display.DataSource=ds.Tables("vTable").DefaultView
 aposted_display.DataBind()
 addshow.visible = true
End Sub
Sub Grid1_Edit(Sender As Object, E As DataGridCommandEventArgs)
 Grid1.EditItemIndex = E.Item.ItemIndex
 ReBind()
End Sub
Sub Grid1_Cancel(Sender As Object, E As DataGridCommandEventArgs)
 Grid1.EditItemIndex = -1
 ReBind()
End Sub
Sub RunSql(vsql as string)
 Dim t As New NameSpaceHersh.SQLService
 Dim s As string
 s = t.RunSQL(ConnStr,vSQL)
 Grid1.EditItemIndex = -1
 Rebind
 if s <> "Success" then
  Message.Text = s
 Message.Style("color") = "red"
 End if
End Sub
Sub hidePanel()
 if AddPanel.visible = true then
  AddPanel.visible = false
```

'reset values adate.text = "" aref.text = "" adr_amount.text = "" acr_amount.text = "" anarr.text = "" end if End Sub Sub UpdateSelection(vselection) sql = "delete from tblSelection) sql = sql + " insert into tblSelection(selection)" sql = sql + " values(" + vselection + ")" runSql(sql)

End Sub

End Class

Chapter 22: Incorporating Web Services in the Trial Balance

Overview

In this chapter, I will modify the TrialBalance.aspx web form built in <u>Chapter 17</u> (The Trial Balance Report) so that it can use the SQLService service. I need to modify the ReBind function that was used to bind two DataGrid controls to two DataView controls. Here is the code listing of the original function that I will modify:

The Original ReBind Function

Sub ReBind()

sql = "SELECT code_display, closing, "

sql = sql + " dr_amount = CASE type WHEN 'A' THEN "

sql = sql + " closing WHEN 'E' THEN closing ELSE 0 END, "

sql = sql + " cr_amount = CASE type WHEN 'l' "

sql = sql + " THEN closing WHEN 'L' THEN closing ELSE 0 END "

sql = sql + " From Masters"

myCommand = New OleDbDataAdapter(SQL, myConnection)
myCommand.Fill(ds, "Masters")
'Binding a Grid
Grid1.DataSource=ds.Tables("Masters").DefaultView
Grid1.DataBind()
'totals
sql = "SELECT 'Total' as nothing ,"
sql = sql + " (Select sum(closing) From masters "
sql = sql + " where type in('A','E')) as dr_total , "
sql = sql + " (Select sum(closing) From masters "
sql = sql + " where type in('I','L')) as cr_total "
myCommand = New OleDbDataAdapter(SQL, myConnection)
myCommand.Fill(ds, "totals")
'Binding a Grid

Grid2.DataSource=ds.Tables("totals").DefaultView

Grid2.DataBind()

End Sub

To use the SQLService service, I have changed this to the following: **The Modified ReBind() Function**

Sub ReBind()

Dim t As New NameSpaceHersh.SQLService

Dim ds As DataSet

sql = "SELECT code_display, closing, "

sql = sql + " dr_amount = CASE type WHEN 'A' THEN "

sql = sql + " closing WHEN 'E' THEN closing ELSE 0 END, "

sql = sql + " cr_amount = CASE type WHEN 'l' "

sql = sql + " THEN closing WHEN 'L' THEN closing ELSE 0 END "

sql = sql + " From Masters"

ds = t.Populate(ConnStr, SQL)

Grid1.DataSource=ds.Tables("vTable").DefaultView

Grid1.DataBind()

'totals

sql = "SELECT 'Total' as nothing ,"

sql = sql + " (Select sum(closing) From masters "

sql = sql + " where type in('A','E')) as dr_total , "

sql = sql + " (Select sum(closing) From masters "

sql = sql + " where type in('l','L')) as cr_total "

ds = t.Populate(ConnStr, sql)

Grid2.DataSource=ds.Tables("vTable").DefaultView

Grid2.DataBind()

End Sub

I call the web service method Populate twice, each time passing the connection string and the SQL query. A DataSet containing the database rows is returned from the function, which I use to bind the two DataGrid controls.

Here is the complete listing of the revised TrialBalance.aspx: **The Modified TrialBalance.aspx**

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<%@ Register TagPrefix="Hersh" TagName="nav" Src="nav.ascx" %>

<html>

<script language="VB" runat="server">

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

if NOT (isPostBack)

rebind

end if

End Sub

Sub ReBind()

Dim t As New NameSpaceHersh.SQLService

Dim ds As DataSet

sql = "SELECT code_display, closing, "

sql = sql + " dr_amount = CASE type WHEN 'A' THEN "

sql = sql + " closing WHEN 'E' THEN closing ELSE 0 END, "

sql = sql + " cr_amount = CASE type WHEN 'l' "

sql = sql + " THEN closing WHEN 'L' THEN closing ELSE 0 END "

```
sql = sql + " From Masters"
```

```
ds = t.Populate(ConnStr, SQL)
```

Grid1.DataSource=ds.Tables("vTable").DefaultView

Grid1.DataBind()

'totals

```
sql = "SELECT 'Total' as nothing ,"
```

sql = sql + " (Select sum(closing) From masters "

sql = sql + " where type in('A','E')) as dr_total , "

sql = sql + " (Select sum(closing) From masters "

```
sql = sql + " where type in('l','L')) as cr_total "
```

```
ds = t.Populate(ConnStr, sql)
```

Grid2.DataSource=ds.Tables("vTable").DefaultView

Grid2.DataBind()

End Sub

</script>

<head>

<style>

a { color:black;

text-decoration:none;}

a:hover { color:red;

text-decoration:underline;}

</style>

</head>

<body style="font: 10pt verdana; background-color:ivory">

<!----- Navigation Start------->

<Hersh:nav id="menu" runat = server

vGridlines = Both

vBorderColor = "Black"

vCellPadding = 7 />

<h3>Trial Balance </h3>

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt">

<Columns>

<asp:BoundColumn HeaderText="Account" DataField="code_display" >

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Debit Amount" DataField="dr_amount">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Credit Amount" DataField="cr_amount">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

</Columns>

<HeaderStyle BackColor="teal" ForeColor="white" Font-Bold="true">

</HeaderStyle>

<ItemStyle ForeColor="DarkSlateBlue">

</ltemStyle>

<AlternatingItemStyle BackColor="Beige">

</AlternatingItemStyle>

</asp:DataGrid>

<!----->

<asp:DataGrid id="Grid2" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt">

<Columns>

<asp:BoundColumn HeaderText="" DataField="nothing">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

</Columns>

<Columns>

<asp:BoundColumn HeaderText="" DataField="dr_total">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="" DataField="cr_total">

<HeaderStyle Width="150px">

</HeaderStyle>

</asp:BoundColumn>

</Columns>

<ItemStyle BackColor="teal" foreColor="white" Font-Bold="true">

</ltemStyle>

</asp:DataGrid>

</body>

</html>

Project 2 Summary

Running an accounting application from a browser has many advantages. I can easily access financial information from anywhere. I do not have to set up network infrastructure to enable various accounting modules to talk to each other. The drawback of a browser-based application is that I am not able to provide a rich GUI interface that users have come to expect from a desktop application. However, if I encapsulate my database access and manipulation routines in a web service, I can use development languages such as Visual Basic.NET or C# to develop the GUI front end and use web services to interact with the database over the Internet. In this project, I showed you how to do this. I developed a web service to interact with the database and modified the Personal Finance Manager to use it. Because this book is on ASP.NET, we used this language as our development language. You could have easily used Visual Basic.NET or C# to develop the user interface and used the web service developed in this project to interact with the database.

Project 3: Inventory Management System

Chapter List

- Chapter 23: The Design of the Inventory Management System
- <u>Chapter 24:</u> Inventory Masters
- <u>Chapter 25:</u> Inventory Movements
- <u>Chapter 26:</u> The Inventory Balances Report

Project 3 Overview

An Inventory Management System enables you to record movements in inventory. Inventory movements are of two basic types: movements inward and movements outward. Inventory purchases and returns from customers or from the shop floor (in the case of manufacturing companies) are examples of movements inward whereas sales, returns to suppliers, or issues to the shop floor (in the case of manufacturing companies) are examples of movements outward.

Good inventory management is one of the most essential functions in an organization. Effective inventory management entails maintaining just the right inventory balance in stock because overstocking can tie up scarce working capital in inventory, and understocking can lead to lost sales. The Inventory Management System provides management data pertaining to movements of inventory and stock balances at any given time.

Traditionally, Inventory Management Systems have been client/server-based. As in the Personal Finance Management system discussed in <u>Part I</u>, we will use web services and ASP.NET to develop a Web-enabled Inventory Management System.

Chapter 23: The Design of the Inventory Management System

Overview

You need to create an MS SQL Server database called ASP.NET and execute the provided file Create.sql (this file is included in the database folder on the book's Web site at <u>www.premierpublishingbooks.com/asp</u>) using the SQL Query Manager (isql) utility of MS SQL Server. <u>Appendix A</u> outlines the steps required to accomplish this.

The Inventory Management System requires the following database objects: **Tables**

- 1. stock_master
- 2. stock_detail
- 3. tr_header

Triggers

- 1. update_stk: Update trigger on stock_detail
- 2. insert_stk: Insert trigger on stock_detail
- 3. delete_stk: Delete trigger on stock_detail

Stored Procedures

- 1. p_stock_masters
- 2. p_stock_trans

The Inventory Masters Table

An inventory master account must be created for each inventory item that you want to use. Each account has a closing field. This field holds the closing balance of that inventory item at any given time. This field is automatically updated by triggers on the stock_detail table. You use this field for displaying the closing balances in reports. Table 23.1 provides the definition of the inventory masters table.

Table 23.1 The stock_master Table

Column	Туре	Length	Description
code_value	integer		ldentity, Primary value
code_display	char	30	Descriptive name
rate	money		
uom	varchar	10	Unit of measureme nt
closing	money		Closing balance
opening	money		Opening

Table 23.1 The stock_master Tab	le		
Column	Туре	Length	Description
			balance

The Transactions Header Table

The tr_header table records the "header" information for an inventory transactions. This is information like date, narration, document number, and so on. The primary key of the tr_header is the document number (doc_no). Note that the Inventory Management System shares the tr_header table with the Personal Finance Manager. If we build an invoicing module, we would need to enter transactions in both the transactions table and the stock_detail table. For example, when recording a sale we would need to record a credit to the sale account, a debit to a bank account, and also record a stock "out" movement. In this case we would have a single tr_header record and have multiple entries in the transactions and stock_detail table that are all tied to the tr_header record based on a unique document number (doc_no). Table 23.2 defines the tr_header table.

		-	
Column	Туре	Length	Description
Id	char	3	Voucher type ("Bank", "Sales", "Purchases")
date	datetime		Voucher date
doc_no	int		Primary key
narr	varchar	150	Narration
ref	varchar	15	Reference

Table 23.2 The tr header Table

The stock_detail Table

This stock movement information will be stored in the $stock_detail$ table. You will record inventory additions and depletions in this table. The primary key of this table is the document number and the serial number (doc_no + sn). Table 23.3 provides a description of the $stock_detail$ table.

Table 23.3 The stock_detail	/ Table		
Column	Туре	Length	Description
doc_no	integer		Primary key
sr_no	money		Primary key
code_value	integer		Inventory account
qty_in	money		Quantity in
qty_out	money		Quantity out

The tr_header and the stock_detail tables are related through the doc_no field in each table. There exists a one-to-many relationship between the two tables. The

stock_detail table has an insert, an update, and a delete trigger defined. The
purpose of these triggers is to update the closing balance field of the stock_master
table after every insert, update, or deletion. I will be discussing these triggers in Chapter
25.

The stored procedures <code>p_stock_masters</code> and <code>p_stock_trans</code> are responsible for inserting and updating inventory master and detail records. I will discuss <code>p_stock_masters</code> in <u>Chapter 24</u> (Inventory Masters) and <code>p_stock_trans</code> in <u>Chapter 25</u>.

Supporting Components

In this project, I will use various components built in earlier chapters of this book. In particular, I will use the database web service developed in <u>Chapter 18</u> and the navigation system developed in <u>Chapter 6</u> and used in <u>Chapter 19</u>.

The SQLService web service component is available in the\SQLService sub-folder of the <u>Project 3</u> samples folder on the book's Web site at

<u>www.premierpress.com/downloads.asp</u>. To install the web service, perform the following steps:

- Create the WSDL file by opening SqlService.asmx so that it goes through IIS (such as http://localhost/your virtual directory/SqlService. asmx). Click on "Service Description." Save the resultant file as SQLService.sdl. Note that you can also do the same thing by browsing to http://localhost/your virtual directory/SqlService.asmx?wsdl.
- Run mSqlproxy.bat. This should put SQLService.dll in the bin directory (and SQlService.vb in the current directory). Remember to modify the "outdir" variable in the bat file to point to your bin folder.
- 3. Run the web form SQLService.aspx and test the services.

The samples folder of this part contains three files which relate to the Navigation component. These files are:

- 1. nav.ascx
- 2. nav.xml
- 3. navigation.aspx

nav.ascx is the user control file. Nav.xml is an xml file that contains the site links for this application and navigation.aspx is web form that is provided so that you can test out this user control.

The site links included in the file nav.xml are as follows:

Nav.xml

<Siteinfo>

<site>

<sitename>Home</sitename>

<siteurl>default.aspx</siteurl>

</site>

<site>

<sitename>Inventory Masters</sitename>

<siteurl>Stockmasters.aspx</siteurl>

</site>

<site>

<sitename>Inventory Transactions</sitename>

<siteurl>stockTrans.aspx</siteurl>

</site>

<site>

<sitename>Inventory Balances</sitename>

<siteurl>Stockbalances.aspx</siteurl>

</site>

</Siteinfo>

Figure 23.1 shows the database schema for this application.



Figure 23.1: The database schema for the Inventory Management System.

Chapter 24: Inventory Masters

The stock_master table stores the master accounts for the inventory system. This table holds information such as inventory item name, rate, unit of measure, opening balance, and closing balance. The web form that I build in this chapter is the interface between the user and the database, and it enables one to insert, update, and delete records from the stock_master table. I use a DataGrid to list the database records. The insert functionality is implemented by adding a number of textboxes on a panel residing on the web form. In both the Edit and Insert modes, the DataGrid calls a database stored procedure p_stock_masters and passes it the field values as input parameters.

Stored Procedure p_stock_masters

The stored procedure p_stock_masters is called from the DataGrid and receives all the field values as input parameters. This procedure handles both the insert and update functionality. The code_value is the primary key of the stock_master table. If a null code_value is passed to the procedure, it inserts a new stock record with the passed parameters; otherwise, it updates the stock record identified by the passed primary key. The following is the complete listing of p_stock_masters:

Stored Procedure p_stock_masters

create procedure p stock master

@code_value integer = null,

@code display varchar(30),

@rate money = 0,

@uom char(10),

@opening money = 0,

```
@closing money =0
```

as

/*

This procedures creates or updates a new stock master record.

If a null code_value is passed, a record is inserted

else the record is updated.

example:

Exec p_stock_master

@code_value = null,

@code_display = "Lux Soap",

@rate = 2,

@uom ="pcs",

```
@opening = 0,
@closing =0
```

*/

DECLARE @flag integer

IF isnull(@code_value,0) = 0

--INSERT--

BEGIN

Insert into stock_master(code_display,rate,uom,opening,closing)

Values(@code_display,@rate,@uom,isnull(@opening,0),isnull(@closing,0))

IF @@ERROR != 0

Begin

GOTO doerror

End

END

ELSE

```
--UPDATE___
```

BEGIN

Update stock_master

Set code_display = @code_display,

rate = @rate,

uom = @uom,

opening =@opening,

closing =@closing

Where code_value =@code_value

IF @@ERROR != 0

Begin

GOTO doerror

End

END

SELECT 0

GOTO doreturn

doerror:

Return - 100

doreturn:

RETURN 0

GO

This stored procedure accepts four input parameters. These correspond to the number of columns displayed by the DataGrid. The DataGrid just gathers the information the user enters and passes it on to this stored procedure. Delegating database interaction to a stored procedure has a number of advantages over executing a SQL query directly from a DataGrid. The stored procedure resides in the database as compiled code and is thus more efficient than a query. I can use temporary tables to simplify my scripting logic. Stored procedures enable me to update multiple tables with ease. The most important advantage is that it completely encapsulates the insert and update logic. If I need to modify this logic, I do not need to change any code in the DataGrid, thus the code maintenance becomes easier. This stored procedure has two modes: Insert and Update. The procedure first checks for the code_value. If the code_value passed to it is Null, it builds an insert statement with the passed parameters. Otherwise, it updates the record having the primary key equal to the passed code_value.

The Inventory Masters Web Form

The Inventory Masters web form is the form that adds, updates, and deletes rows to the $stock_masters$ table. This form makes a call to the stored procedure $p_stock_masters$ to insert or update inventory records. Figure 24.1 shows what this looks like.

Add Account Add Account Call Instant Call Instant	nventory				
Add Account Add Scount Collection of a finance Add Scount Total bodies Inter Section of a finance Solid bodies Counted of a finance Solid bodies Counted of a finance Solid bodies Counted of a finance Solid bodies Solid bodies Solid bodies Solid bodies Solid bodies Solid bodies	inventory i				
Add Actourd Call Colored Actourd & Name Sector Sector Se		Masters			
Add Control & Add and Add and Add Add Add Control (Control (Contro	Add Account				
ofer (Indexe) is a linear trans stag) is jess 0 0 Edit (Indexe) is Consent 0 1 Edit (Inte 0 0 Edit (Indexe) is Advasars & Indexess Stage (I-1) (Inte 0 0 Edit (Indexe) is Advasars & Indexess Stage (I-1) (Inte 0 0 I Z	7 1000000000000000000000000000000000000	The second se		and a second second	
214 Dedee 1 Count Ol 10 75 0 0 Dit Delev 2 Advant & Norsen Son 0 0 13 00 0 0 12	Idit Delete 4	how Tone aciab	2 010 0	and the second s	
titi [Sales 2 Julean & Mercan Song 9.5 0 0 12	Deleter 1	Countered Oil	100 Tam 0		
	Nill Delate 2	Johnson & Johnson Scian	9.8 Jun 0		
				3.2	
				1.2	

Figure 24.1: The Inventory Masters web form. Figure 24.2 shows the Inventory Masters web form in Add mode. Figure 24.3 shows the Inventory Masters web form in Edit mode.

Breatway Masters Breatway Transmission Breatway Bulances Masters Internet Masters Internet Masters Markers Internet Masters Internet Masters		2 vention	Inventory Bulances	
Aborg Masters		atory Master	Add a Year Account Asma: Rem: Unit Comma Value Scheid	0.001 0.001 <th< th=""></th<>
must not and State Market have a state of the state of th	Inter 24.2.1 The Inventory Massers web form in the Add in Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors) Inter Franker Side web (Sectors)			
	Motion Masters Motion Motion Rame Non Relating Classing Motion Non Relating Classing Classing Motion Non Relating Classing Classing Station Non Relating Classing Classing Station Non Relating Classing Classing Station Non Relating Relating Classing Station Non Relating Relating Relating	ure 24	ers web form in the Add n	The Investor One of the Second
Investory Radeo Directory Protocology Sensitive Malanae	District Annual Balan Name District District Closelog of Solids k Solids Name District District Closelog of Solids k Solids Name District District District Solids k Solids Name District District District Solids k Solids Name Name District District Solids k Solids Name Name District District		ers web form in the Add n	The Inventor Masters
Directory Robert Directory Transitions Severality Educes	During Occurs Name Rate Name During Classing of Service k Lar.Scop P Final P	Theorem is the office of the o	ers web form in the Add m	The Investigation of the second strength of t
Directory Hates Directory Transition Directory Educations Intory Masters Internet	m ² Code 1 Europo 0 01 01 00 01 00 01 00 00 00 00 00 00 0	These weeks	ers web form in the Add m	The Investigation of the second seco
Divertiery Plantes Divertiery Transactions Sensitives Extenses Internet Masters Notation Notation Notation Sensity Referent Notation Notation Notation	Delate 2 Interest & Determining 45 det ID	The set of	ers web form in the Add n	The Investigation of the second seco
Breaters Nations Breaters Transmissions Breaters Educates Notations Account		Thread I and a second s	ers web form in the Add n	The Investigation of the second seco

Bank Stream and the Stream Stream Non-tell Decision Stream Strea Figure 24.3: The Inventory Masters web form in the Edit mode.

The web form StockMasters.aspx, and its Code-Behind form StockMasters.vb, contain the required code. The site navigation for this project is handled by the navigation user control discussed in Chapter 6, "User Controls." It is registered at the top of the StockMasters.aspx web page as follows:

<%@ Register TagPrefix="Hersh" TagName="nav" Src="nav.ascx" %>

This user control is initiated within the Web page as follows:

<!----- Navigation Start-----> <Hersh:nav id="menu" runat = server vGridlines = Both vBorderColor = "Black"

vCellPadding = 7 />

<!----- Navigation Ends----->
A DataGrid on the web form handles the display and editing of the records from the
stock_master table as follows:

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

OnEditCommand="Grid1_Edit"

OnCancelCommand="Grid1_Cancel"

OnUpdateCommand="Grid1_Update"

OnDeleteCommand = "Grid1_delete"

DataKeyField="code_value"

AllowPaging="True"

PageSize="20"

PagerStyle-Mode="NumericPages"

PagerStyle-HorizontalAlign="Right"

PagerStyle-NextPageText="Next"

PagerStyle-PrevPageText="Prev"

OnPageIndexChanged="MyDataGrid_Page"

AllowSorting="true"

OnSortCommand="MyDataGrid_Sort"

<Columns>

<asp:EditCommandColumn

EditText="Edit"

CancelText="Cancel"

UpdateText="OK"

ItemStyle-Wrap="false"

HeaderText="Edit"

HeaderStyle-Wrap="false"/>

<asp:ButtonColumn Text="Delete" CommandName="Delete"

HeaderText="Delete"/>

<asp:BoundColumn HeaderText="Account #" ReadOnly="true"

DataField="code_value"

SortExpression="code_value" />

<asp:TemplateColumn HeaderText="Name"

SortExpression="code_display">

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("code_display") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:RequiredFieldValidator runat=server

controltovalidate=edit_Name

errormessage="Name is required.">*

</asp:RequiredFieldValidator>

<asp:TextBox id="edit_name"

Text='<%# Container.DataItem("code_display") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Rate" SortExpression="rate" >

<ltemTemplate>

<asp:Label Text='<%# Container.DataItem("rate") %>' runat="server" />

</ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_rate" Text='<%# Container.DataItem("rate") %>' runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="uom" SortExpression="uom" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("uom") %>' runat="server"/> </ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_uom" Text='<%# Container.DataItem("uom") %>' runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Opening" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("opening") %>' runat="server"/> </ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_opening"

Text='<%# Container.DataItem("opening") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Closing" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("closing") %>' runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_closing" BorderStyle="None" Readonly="True"

Text='<%# Container.DataItem("closing") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

</Columns>

<HeaderStyle BackColor="Gray" ForeColor="White" Font-Bold="true"/>

<ItemStyle ForeColor="DarkSlateBlue"/>

<AlternatingItemStyle BackColor="Beige"/>

</asp:DataGrid>

The EditCommandColumn is the ASP.NET generated column, which creates the Edit, OK, and Cancel link buttons. It is created as follows:

<asp:EditCommandColumn

EditText="Edit"

CancelText="Cancel"

UpdateText="OK"

ItemStyle-Wrap="false"

HeaderText="Edit"

HeaderStyle-Wrap="false"/>

I have specified three commands in the DataGrid that correspond to these buttons. These are the following:

OnEditCommand="Grid1_Edit"

OnCancelCommand="Grid1_Cancel"

OnUpdateCommand="Grid1_Update"

When the Edit link is clicked, the Grid1_Edit function is fired. This event contains the following code:

Sub Grid1_E dit(Sender As Object, E As DataGridCommandEventArgs)

Grid1.EditItemIndex = E.Item.ItemIndex

ReBind()

End Sub

The DataGrid needs to be told which row is being edited. This is done by setting the EditItemIndex property of the DataGrid to the index of the button that was clicked in the above code snippet. Clicking on the Cancel button fires the Grid1_Cancel function. This function simply sets the EditItemIndex property of the DataGrid to -1, as in the following script:

Sub Grid1_Cancel(Sender As Object, E As DataGridCommandEventArgs)

Grid1.EditItemIndex = -1 ReBind()

End Sub

Note that in both the cases, I call the ReBind function. This function rebinds the DataGrid to the data source so that the DataGrid is updated with the changes that the user makes. The Inventory Master web form makes use of the populate function of the SQLService web service to bind to a DataGrid. This function is passed the SQL Query, and the connection string and a DataSet is received back from it. The default view of this DataSet is used to bind the DataGrid.

The ReBind Function

Sub RunSql(vSQL as string)

Dim ConnStr As String

Dim SQL As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

Dim t As New NameSpaceHersh.SQLService

Dim s As string

s = t.RunSQL(ConnStr,vSQL)

Grid1.EditItemIndex = -1

if s <> "Success" then

Message.Text = s

Message.Style("color") = "red"

end if

response.write (vsql)

Rebind

End Sub

The Sub <u>Grid1_update</u> handles the Update logic. This function is fired when the grid is in the Edit mode. A SQL query string is built dynamically, which calls the p_stock_masters stored procedure with a document number. (Remember, we call the procedure with a null document number to effect an insert and pass it a valid document number to effect an update.)

Grid1_update

Sub Grid1_Update(sender As Object, e As DataGridCommandEventArgs)

Dim sql As string

Dim code_display As String

Dim rate As String Dim uom As String Dim opening As String Dim closing As String Dim myTextBox As TextBox 'This is the key value: 'Retrieved from the DataKey, since it's a read only field Dim code_value as string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString myTextBox = E.Item.FindControl("edit_name") code_display = mytextbox.text myTextBox = E.Item.FindControl("edit_rate") rate = mytextbox.text myTextBox = E.Item.FindControl("edit_uom") uom = mytextbox.text myTextBox = E.Item.FindControl("edit_opening") opening = mytextbox.text myTextBox = E.Item.FindControl("edit_closing") closing = mytextbox.text 'Now execute stored procedure sql = "Execute p_Stock_master @code_value =" sql = sql+code_value+", @code_display = "+code_display+"',@rate=" sql = sql+rate+", @uom='"+uom +"', @opening ="+opening+", @closing="+closing RunSql(sql)

End Sub

This SQL string is passed on to the function RunSQL that does the actual work of executing the SQL statement. Note that I extract the primary key (code_value) and pass it on to the procedure. The existence of a valid code_value tells the procedure to issue an update statement. If you pass it a null code_value, it will issue an insert statement.

Adding Records

Three textboxes and one button have been added to the form. These controls reside on a panel with the id AddPanel. In the aspx form, I have added HTML comments to show where the section begins and ends.

Insert Logic in the Form

<!----> insert row logic----->

```
<asp:Panel id="AddPanel" runat="server" Visible="false">
```

Add a New Account:

Name:

<asp:TextBox id="acode_display" runat="server" />

<asp:RequiredFieldValidator runat=server

controltovalidate=acode_display

errormessage="Name is required.">*

</asp:RequiredFieldValidator>

Rate:

<asp:TextBox id="arate" runat="server" />

Unit:

<asp:TextBox id="auom" runat="server" />

Opening Value:

<asp:TextBox id="aopening" value = "0" runat="server" />

```
<asp:Button id="SubmitDetailsBtn" text="Submit"
onclick="add_Click" runat="server" />

</asp:Panel>
<!------Insert Logic ends ------>
```

A button (id = AddShow) displays with the caption Add Account on the web form. Clicking on this button fires the add_show Sub. This Sub simply sets the visible property of the panel to true. Once the panel is visible, all the controls on the panel also become visible. At this point, all the textboxes are ready for accepting user input. The insert logic is handled by the function add_click. It builds a SQL string by extracting the text properties of various textboxes. The following is the Sub: The add_click Sub

Sub add_click(Source As Object, E As EventArgs)

Dim sql As string

if acode_display.text = "" then

response.write("Incomplete information")

exit sub

end if

SQL = "Execute p_stock_master @code_value=NULL,@code_display='"

SQL = SQL+ acode_display.text + "',@rate="

SQL = SQL+arate.Text+",@uom="+auom.text+",@opening="+aopening.text

RunSql(sql)

'reset values

acode_display.text = ""

aopening.text = ""

arate.text = ""

auom.text = ""

hidePanel()

End Sub

Note that we are passing a NULL code_value to the procedure. This fires an insert statement. This SQL query string is passed on to the function RunSQL, which does the actual work of executing the SQL statement.

Delete Mode

The Delete mode is activated when the user clicks on the delete link. I have created a delete ButtonColumn as follows:

<asp:ButtonColumn Text="Delete" CommandName="Delete" HeaderText="Delete"/> I have associated an OnDeleteCommand with this button as follows:

<asp:DataGrid id="Grid1" runat="server"

OnDeleteCommand = "Grid1_delete"

----- >

In the Grid1_Delete event, I simply build a delete statement and pass it on to the function RunSQL, which applies the query against the database.

Sub Grid1_delete

Sub Grid1_delete(sender As Object, e As DataGridCommandEventArgs)

Dim code_value As string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString

Dim sql As string

sql = "Delete from stock_master where code_value = " + cstr(code_value)

RunSql(sql)

End Sub

The RunSql Function

This is a generic function, which executes a SQL Action statement against the database. The SQL query statement is passed to it as a parameter. The <u>Gridl_update</u> Sub, the Add_click Sub, and the Gridl_delete Sub call this function to update, add, or delete a record. This function calls the RunSql function of the web service and passes it the connection string, as well as the SQL action query/procedure call. If the procedure/query was executed successfully, the string Success is returned from the function. Otherwise, the appropriate error string is returned, which is displayed in the browser.

Sub RunSql

Sub RunSql(vSQL as string)

Dim ConnStr As String

Dim SQL As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

Dim t As New NameSpaceHersh.SQLService

Dim s As string

s = t.RunSQL(ConnStr,vSQL)

Grid1.EditItemIndex = -1

if s <> "Success" then

Message.Text = s

Message.Style("color") = "red"

End if

response.write (vsql)

Rebind

End Sub

Sorting

The DataGrid enables you to sort the columns by clicking on a link below the columns. Setting the AllowSorting property to true triggers this built-in mechanism, as follows:

<asp:DataGrid id="Grid1" runat="server"

AllowSorting="true"

OnSortCommand="MyDataGrid_Sort">

When this property is set to true, the DataGrid renders the column captions with a linkbutton. If you now click on a column, the OnSortEvent is fired. This event contains the following code:

Sub MyDataGrid_Sort(sender As Object, e As DataGridSortCommandEventArgs)

SortField = e.SortField

ReBind

End Sub

The SortField variable is a Public (string) variable, which holds the name of the column, and by which the DataGrid is to be sorted. It is initially set to the code_display column in the page_load event as follows:

Sub Page_Load(Source As Object, E As EventArgs)

If NOT (isPostBack)

If SortField = "" Then

SortField = "code_display"

End If ReBind

End If

End Sub

You need to set the SortExpression attribute in the column templates. For example, for the rate column to participate in sorting, you have to set the template as follows:

<asp:TemplateColumn HeaderText="Rate" SortExpression ="rate" >

</asp:TemplateColumn>

Each time the user clicks on a "sortable" column, the MyDataGrid_Sort Sub is fired. The SortField variable again gets set in this method.

Sub MyDataGrid_Sort(sender As Object, e As DataGridSortCommandEventArgs)

SortField = e.SortExpression

ReBind

End Sub

The ReBind function uses the SortExpression attribute to sort the DataView (which in turn binds the DataGrid), and refreshes the DataGrid to reflect the rows sorted by the new sort field, as follows:

Sub ReBind()

Dim ConnStr As String

Dim SQL As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

Dim t As New NameSpaceHersh.SQLService

Dim ds As DataSet

' Bind Grid

SQL = "Select * from stock_master"

ds = t.Populate(ConnStr, SQL)

```
Dim dv2 As DataView
```

dv2 = ds.Tables("vTable").DefaultView

```
dv2.Sort = SortField
```

Grid1.DataSource= dv2

```
Grid1.DataBind()
```

hidePanel()

End Sub

Paging in DataGrid

The DataGrid has a built-in pager control, which displays a user-defined number of pages per page, and also numeric or next/previous buttons at the bottom of the DataGrid. Clicking on these links displays the next set of pages. To enable paging, you set a number of properties as follows:

<asp:DataGrid id="Grid1" runat="server"

```
AllowPaging="True"
```

PageSize="3"

PagerStyle-Mode="NumericPages"

PagerStyle-HorizontalAlign="Right"

PagerStyle-NextPageText="Next"

PagerStyle-PrevPageText="Prev"

OnPageIndexChanged="MyDataGrid_Page">

The AllowPaging property must be set to true to enable paging. The PageSize property sets the number of records per page. A PageSize of 3 implies that only three records per page will be shown. If you leave out the PagerStyle-Mode="NumericPages" property, then instead of numeric links at the bottom, you get two links: next and previous. The PagerStyle-NextPageText and the

PagerStyle-PrevPageText properties are descriptive captions for these two links and they can be any text you want. You are required to code one event. This is the OnPageIndexChanged event, which fires off the MyDataGrid_Page event. You simply call the ReBind function in this event as follows:

Sub MyDataGrid_Page(sender As Object, e As DataGridPageChangedEventArgs)

ReBind

End Sub

Here is the complete listing of StockMaster.aspx. **StockMaster.aspx**

<%@Page Language="VB" Inherits="BaseClass" Src="stockmasters.vb" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<%@ Register TagPrefix="Hersh" TagName="nav" Src="nav.ascx" %>

<html>

<script language="VB" runat="server">

Sub Grid1_Update(sender As Object, e As DataGridCommandEventArgs)

Dim sql As string

Dim code_display As String

Dim rate As String

Dim uom As String

Dim opening As String

Dim closing As String

Dim myTextBox As TextBox

'This is the key value:

'Retrieved from the DataKey, since it's a read only field

Dim code_value As string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString

myTextBox = E.Item.FindControl("edit_name")

code_display = mytextbox.text

myTextBox = E.Item.FindControl("edit_rate")

rate = mytextbox.text

myTextBox = E.Item.FindControl("edit_uom")

uom = mytextbox.text

myTextBox = E.Item.FindControl("edit_opening")

opening = mytextbox.text

myTextBox = E.Item.FindControl("edit_closing")

closing = mytextbox.text

'Now execute stored procedure

sql = "Execute p_Stock_master @code_value ="

sql = sql+code_value+", @code_display = "+code_display+", @rate="

sql = sql+rate+", @uom="+uom +"',@opening ="+opening+",@closing="+closing

RunSql(sql)

End Sub

Sub add_click(Source As Object, E As EventArgs)

Dim sql As string

if acode_display.text = "" then

response.write("Incomplete information")

exit sub

end if

SQL = "Execute p_stock_master @code_value=NULL,@code_display='"

SQL = SQL+ acode_display.text + "',@rate="

SQL = SQL+arate.Text+",@uom="+auom.text+",@opening="+aopening.text

RunSql(sql)

'reset values

acode_display.text = ""

aopening.text = ""

arate.text = "" auom.text = "" hidePanel() End Sub Sub add_show(Source As Object, E As EventArgs) AddPanel.visible = true End Sub Sub Grid1_delete(sender As Object, e As DataGridCommandEventArgs) Dim code_value As string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString Dim sql As string sql = "Delete from stock_master where code_value = " + cstr(code_value) RunSql(sql) End Sub Sub MyDataGrid_Page(sender As Object, e As DataGridPageChangedEventArgs) Grid1.CurrentPageIndex = e.NewPageIndex ReBind End Sub Sub MyDataGrid_Sort(sender As Object, e As DataGridSortCommandEventArgs) SortField = e.SortExpression ReBind End Sub </script> <head> <style> a { color:black; text-decoration:none;} a:hover { color:red; text-decoration:underline;} </style> </head>

<body style="font: 10pt verdana; background-color:ivory">

<form runat="server">

<asp:ValidationSummary runat=server

headertext="There were errors on the page:" />

<!----- Navigation Start------>

<Hersh:nav id="menu" runat = server

vGridlines = Both

vBorderColor = "Black"

vCellPadding = 7 />

<!----- Navigation Ends------->

<h3>Inventory Masters </h3>

<asp:Label id="Message" runat="server"/>

<asp:Button id="Addshow" text="Add Account" onclick="add_show" runat="server" />

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

OnEditCommand="Grid1_Edit"

OnCancelCommand="Grid1_Cancel"

OnUpdateCommand="Grid1_Update"

OnDeleteCommand = "Grid1_delete"

DataKeyField="code_value"

AllowPaging="True"

PageSize="20"

PagerStyle-Mode="NumericPages"

PagerStyle-HorizontalAlign="Right"

PagerStyle-NextPageText="Next"

PagerStyle-PrevPageText="Prev"

OnPageIndexChanged="MyDataGrid_Page"

AllowSorting="true"

OnSortCommand="MyDataGrid_Sort">

<Columns>

<asp:EditCommandColumn

EditText="Edit"

CancelText="Cancel"

UpdateText="OK"

ItemStyle-Wrap="false"

HeaderText="Edit"

HeaderStyle-Wrap="false"/>

<asp:ButtonColumn Text="Delete" CommandName="Delete"

HeaderText="Delete"/>

<asp:BoundColumn HeaderText="Account #" ReadOnly="true"

DataField="code_value" SortExpression="code_value" />

<asp:TemplateColumn HeaderText="Name"

SortExpression="code_display">

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("code_display") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:RequiredFieldValidator runat=server

controltovalidate=edit_Name

errormessage="Name is required.">*

</asp:RequiredFieldValidator>

<asp:TextBox id="edit_name"

Text='<%# Container.DataItem("code_display") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Rate" SortExpression="rate" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("rate") %>' runat="server" />

</ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_rate" Text='<%# Container.DataItem("rate") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="uom" SortExpression="uom" >

<ltemTemplate>

<asp:Label Text='<%# Container.DataItem("uom") %>' runat="server"/>

</ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_uom" Text='<%# Container.DataItem("uom") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Opening" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("opening") %>' runat="server"/>

</ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_opening"

Text='<%# Container.DataItem("opening") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Closing" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("closing") %>'

runat="server"/>

</ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_closing" BorderStyle="None" Readonly="True"

Text='<%# Container.DataItem("closing") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

</Columns>

<HeaderStyle BackColor="Gray" ForeColor="White" Font-Bold="true"/>

<ItemStyle ForeColor="DarkSlateBlue"/>

<AlternatingItemStyle BackColor="Beige"/>

</asp:DataGrid>

<!---- insert row logic----->

<asp:Panel id="AddPanel" runat="server" Visible="false">

Add a New Account:

Name:

<asp:TextBox id="acode_display" runat="server" />
<asp:RequiredFieldValidator runat=server

controltovalidate=acode_display

errormessage="Name is required.">*

</asp:RequiredFieldValidator>

Rate:

<asp:TextBox id="arate" value = "0" runat="server" />

Unit:

<asp:TextBox id="auom" value = "pcs" runat="server" />

Opening Value:

<asp:TextBox id="aopening" value = "0" runat="server" />

<asp:Button id="SubmitDetailsBtn" text="Submit"

onclick="add_Click" runat="server" />

</asp:Panel>

<!-----> Insert Logic ends

</form>

</body>

</html>

Stock_masters.vb is the Code Behind file. StockMasters.vb

Option Strict Off

Imports System

Imports System.Collections

Imports System.Text

Imports System.Data

Imports System.Data.OleDb

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class BaseClass

Inherits System.Web.UI.Page

Protected Grid1 as DataGrid

Protected Message as label

Protected AddPanel as Panel

Public SortField As String

Sub Page_Load(Source As Object, E As EventArgs)

If NOT (isPostBack)

If SortField = "" Then

SortField = "code_display"

End If

rebind

End If

End Sub

Sub ReBind()

Dim ConnStr As String

Dim SQL As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

Dim t As New NameSpaceHersh.SQLService

Dim ds As DataSet

' Bind Grid

SQL = "Select * from stock_master"

ds = t.Populate(ConnStr, SQL)

Dim dv2 As DataView

dv2 = ds.Tables("vTable").DefaultView

dv2.Sort = SortField

Grid1.DataSource= dv2

Grid1.DataBind()

hidePanel()

End Sub

Sub Grid1_Edit(Sender As Object, E As DataGridCommandEventArgs)

Grid1.EditItemIndex = E.Item.ItemIndex

ReBind()

End Sub

Sub Grid1_Cancel(Sender As Object, E As DataGridCommandEventArgs)

Grid1.EditItemIndex = -1

ReBind()

End Sub

Sub hidePanel()

if AddPanel.visible = true then

AddPanel.visible = false

End if

End Sub

Sub RunSql(vSQL as string)

Dim ConnStr As String

Dim SQL As String

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

Dim t As New NameSpaceHersh.SQLService

Dim s As string

```
s = t.RunSQL(ConnStr,vSQL)
```

Grid1.EditItemIndex = -1

if s <> "Success" then

Message.Text = s

Message.Style("color") = "red"

End if

response.write (vsql)

Rebind

End Sub

End Class

Chapter 25: Inventory Movements

The Inventory Management System records movements of inventory items. You can add to the inventory by purchasing stock, by inventory returns from customers, by return of unused materials from the shop floor (in the case of manufacturing units), and so on. You can reduce inventory by sales, issues to shop floor, supplier returns, and so on. The web form that we will build in this chapter will be responsible for recording stock additions and depletions. Each stock movement (addition or depletion) has a "header" entry, which is recorded in the tr_header table and a "detail" entry, which is recorded in the tr_header table and a "detail" entry, which is recorded in the stock_detail table. A DataGrid on the web form lists all available movements and also allows editing of any particular movement. A number of TextBox controls residing on a panel on the form provide the functionality to add stock movements. Both the Insert and the Update modes gather the user inputs and pass them on to the stored procedure p_stock_trans, which encapsulates the insert and update logic.

Inserting and Updating Transactions

The stored procedure p_stock_trans handles both the insert and update logic. It has seven input parameters that are passed to it by the DataGrid or the record addition form. To insert a new inventory movement, you pass a null document number (doc_no) to this procedure and to modify an existing movement, you pass it the document number of the inventory movement.

Stored Procedure p_stock_trans

create procedure p_stock_trans

@date datetime ,

@ref varchar(30) = NULL,

@qty_in money = 0,

@qty_out money =0,

@id char(3),

@doc_no integer = NULL,

@narr varchar(150) = NULL,

@code_value integer

as

/*

call with a null doc_no to insert, a valid doc_no to update

Example :

Execute p_stock_trans

@date =getdate() ,

@ref = test1,

```
@qty_in = 10,
```

@qty_out =0,

```
@id ="STK",
```

```
@doc_no = NULL,
```

@narr = "Test Entry"

*/

```
DECLARE @II_doc integer
```

DECLARE @ret integer

BEGIN TRANSACTION

--To insert a new movement, pass doc_no =NULL to procedure--

IF isnull($@doc_no,0$) = 0

BEGIN

--SafeGuard: Check if tranaction with same ref# exists.

--If so dont insert

select @ret = count(*) from tr_header where ref = @ref

if @ret > 0

BEGIN

GOTO doerror

END

Select @II_doc = isnull(max(doc_no),0)+1 from tr_header

IF @@ERROR != 0

Begin

GOTO doerror

End

END

ELSE

-----To UPDATE pass the doc_no-----

BEGIN

SELECT @II_doc = @doc_no

delete from stock_detail where doc_no = @doc_no

IF @@ERROR != 0

Begin

GOTO doerror

End

delete from tr_header where doc_no = @doc_no

```
IF @@ERROR != 0
```

Begin

GOTO doerror

End

END

BEGIN

INSERT INTO tr_header (id, date,ref,doc_no ,narr) VALUES (@id, isnull(@date,getdate()),@ref,@ll_doc, @narr) IF @@ERROR != 0 Begin GOTO doerror END

INSERT INTO stock_detail (doc_no, qty_in, qty_out, code_value, sr_no)

VALUES (@II_doc, isnull(@qty_in,0), ISNULL(@qty_out,0), @code_value, 1)

IF @@ERROR != 0

Begin

GOTO doerror

End

END

COMMIT TRANSACTION

SELECT 0

GOTO doreturn

doerror:

Rollback TRANSACTION

doreturn:

RETURN 0

SELECT -100

To add (insert) a new record, you will pass a null value to @doc_no parameter of the stored procedure. In this case, the procedure selects the maximum doc_no from the tr_header table, increments it by one, and stores it in the variable @ll_doc. To modify (update) an existing stock detail transaction, you pass an existing document number to the @doc_no parameter of the stored procedure. In this case, the procedure stores the passed document number to the variable @ll_doc. It then deletes the stock_detail records transactions having this doc_no, as they will again be recreated with the passed parameters. This process of deleting and reinserting the

stock_detail records enable the delete and insert triggers associated with the stock_detail table to fire. As will be explained in the <u>next section</u>, this process updates the closing balance field of the appropriate stock_master record. Finally, for both the insert and update modes, a row is created in the tr_header table having a document number equal to the value stored in the variable @ll_doc. A row is also created in the stock_detail table with the passed parameters.

Triggers on the stock_detail Table

The stock_detail table has an insert, an update, and a delete trigger. These triggers update the closing balance field of the stock_master table to reflect the most current stock balance. The closing stock balance is thus current after every stock movement. These triggers are listed below.

The <u>insert_stk</u> trigger is an insert trigger on the stock_detail table. Its code is as follows:

Insert_stk

CREATE TRIGGER insert_stk ON stock_detail for insert as

DECLARE @sql varchar(200)

DECLARE @mtype char(1)

DECLARE @bal money

SELECT *

into #temp

from inserted

BEGIN

SELECT @bal = ISNULL(#temp.qty_in,0) - ISNULL(#temp.qty_out,0)

FROM #temp

END

UPDATE stock_master

SET closing = isnull(closing,0) + @bal

FROM stock_master, #temp

WHERE (stock_master.code_value = #temp.code_value)

The <u>update_stk</u> trigger is an update trigger on the table stock_detail. Its code is as follows:

Update_stk

CREATE TRIGGER update_stk ON stock_detail for update as Declare @sql varchar(200) DECLARE @mtype char(1) DECLARE @bal money SELECT * into #temp from inserted

SELECT *

into #t2

from deleted

BEGIN

SELECT @bal = ISNULL(#temp.qty_in,0) - ISNULL(#temp.qty_out,0)

-(ISNULL(#t2.qty_in,0) - ISNULL(#t2.qty_out,0))

From #temp, #t2

Where #temp.code_value = #t2.code_value

END

UPDATE stock_master

SET closing = isnull(closing,0) + @bal

FROM stock_master, #temp WHERE (stock_master.code_value = #temp.code_value)

The <u>delete_stk</u> trigger is an update trigger on the table stock_detail. Its code is as follows:

delete_stk

CREATE TRIGGER delete_stk ON stock_detail for delete as

DECLARE @sql varchar(200)

DECLARE @mtype char(1)

DECLARE @bal money

SELECT *

into #temp

from deleted

UPDATE stock_master

SET Closing =Closing-

(ISNULL(#temp.qty_in,0) - ISNULL(#temp.qty_out,0))

FROM stock_master, #temp

WHERE (stock_master.code_value = #temp.code_value)

These triggers make use of the Microsoft SQL Server deleted and inserted tables to access the before and after values of the fields. An *inserted table* is a SQL Server table, which holds the inserted values in case of an insert statement, or the updated values in case of an update statement. A *deleted table* is a Microsoft SQL Server table that holds the original values in case of an update statement or the deleted value in case of a delete statement. These tables have the same fields as the table it references, which, in this case, is the stock_detail table. These triggers simply apply an arithmetic formula to arrive at the closing balance. <u>Table 25.1</u> describes the calculations performed by each trigger in calculating the closing balance.

Table 25.1 Closing Balance Calculations				
Action	Trigger	Closing balance formula		
Insert	<u>insert_stk</u>	<pre>closing + (inserted.qty_ in - inserted.qty_o ut)</pre>		
Update	update_stk	<pre>closing + (inserted.qty_ in - inserted.qty_o ut) minus (deleted.qty_i n - deleted.qty_ou t)</pre>		
Delete	delete_stk	<pre>closing - (deleted.qty_i n - deleted.qty_ou t)</pre>		

Inventory Transactions

Inventory Transaction maintenance involves adding, modifying, and deleting transactions. The following objects are involved in this Sub system:

- 1. The Inventory Transactions web form (StockTrans.aspx) and the Code-Behind form (StockTrans.vb)
- The stored procedure (p_stock_trans)

The Inventory Transactions Form

The Inventory Transactions web form enables users to add and modify inventory transaction records. The add functionality is provided by TextBox controls, residing on a panel that is made visible when the Add button is clicked. A DataGrid implements the modify functionality. This form is quite similar to the stock_masters form that we built in <u>Chapter 24</u>, "<u>Inventory Masters.</u>" The DataGrid and the Add portion of the form are set up as described in that chapter.

Figure 25.1 shows what the form looks like.

a second s	constituted in the second statement	the sub-state of the sub-state of the sub-state of the sub-							1000
And a growth (g) the first state of the growth from the set of the						21 9			
a franc	o Deserve	at Contain the							
Form .	Divertory Mad	9-3	Inerter	i Transactio	otie		Secondary Bala	(1)#1	
inventor	y Movemen	nts							
New Tetra	100								
2002000	In Plant Calls	Investory	Accessed in	- extension	() and ()	In Quantity Out			
titit Delete 1	1 2000-15	-RETER OF COLORADA & 3	lahasan Saup P	sectors.	110				
101 Delete 2	1 2000-11	411100-080-031 Julyasan Is 3	Wheen Seat a	aini .	1	10			
Data Detatar 1	1 2000-01	-EXTED OD DO Due Type of	140 7	witness.	1100	1			
								@Sterret.	

Figure 25.1: The Inventory Transactions form.

Figure 25.2 shows the Inventory Transactions Form in Add mode. Figure 25.3 shows the Inventory Transactions Form in Edit mode

Breathers Mades Directory Transmission Desembers Transmission InternetTory Movements Set Directory Movements InternetTory Movements Set Directory Movements Set Directory InternetTory Movements Set Directory Movements Set Directory Directory InternetTory Set Directory Movements Set Directory Directory Directory InternetTory Set Directory Directory Directory Directory Directory InternetTory Set Directory Directory Directory Directory Directory InternetTory Set Directory Directory Directory Directory Directory InternetTory Directory Directory Directory Directory Directory InternetTory Directory Directory Directory Directory Directory
entory Hovements Set
No Terristica - Second register and the second registe
Sales Entry Sales Description Description <thdescrip< th=""> <thdescrip< th=""> Descrip<!--</th--></thdescrip<></thdescrip<>
Delay 1 2 2010 44 (againt) Default is any and the second s
Notes 2 2 2 2010 dig Marce A select 0 20 Access Folded To: Access
Newsfield
attern 2 2 millional do and Purchases 2000 0 Quantity (b)
Quantity Out:
Submit

Figure 25.2: The Inventory Transactions form in Add mode.

nventory Mov	vements					
New Trendston	1					
Edit Balata Det	a.r	O at a	Investory Account:	Namation	Questily 3a	Quantity Dut.
Edit Delete &	h	2000-05-01700-00-00	Johnson & Johnson Scap	Purchases	100	
Di Cancel Juliés 2	2	2000-01-01100-00.00	2	Lates	0	0
duit Delete 3	3	2000-02-01710-00-00	due Tone scep	Furtheres	5000	
fill Delete H	1	2000-00-01700-00-00	Lar trap	Furtheeni	300	e .

Figure 25.3: The Inventory Transactions form in Edit mode.

The ReBind Function

The ReBind function binds the DataGrid to a SQL query, first in the Page_load event, and then whenever the data changes and the Grid needs to be refreshed. Sub ReBind shows the script of the function.

Sub ReBind

Sub ReBind()

Dim t As New NameSpaceHersh.SQLService

Dim ds As DataSet

'DataSetCommand

SQL = "select * from tr_header h, stock_detail s, stock_master m"

SQL = SQL + " where h.doc_no = s.doc_no"

SQL = SQL + " AND s.code_value = m.code_value"

ds = t.Populate(ConnStr, SQL)

Grid1.DataSource=ds.Tables("vTable").DefaultView

Grid1.DataBind()

'populate inventory account(add mode) selection drop down list

SQL = "Select * from stock_master"

ds = t.Populate(ConnStr, SQL)

acode_value.DataSource=ds.Tables("vTable").DefaultView

acode_value.DataBind()

addshow.visible = true

End Sub

In this function, I call the web service method Populate twice, each time passing to it as parameters the connection string and the SQL query. A DataSet containing the result set is returned from the function, which I use to bind the DataGrid and the DropDownList, respectively.

The Add Mode

When the addshow button is clicked, the add_show Sub is fired. This Sub simply sets the visible property of the panel AddPanel to true. This, in turn, makes all the controls residing on this panel visible.

Sub add_show

Sub add_show(Source As Object, E As EventArgs)

AddPanel.visible = true

End Sub

The input controls for the Insert mode have been marked in the web form within the HTML comment blocks Insert Logic Starts and Insert Logic Ends. Each control has an associated id property, which will be used to refer to the control. There is a RequiredFieldValidator attached to the Date and ref fields. The stored procedure p_stock_trans will also check for the uniqueness of the ref field. If it is not unique, the procedure will return an error condition. The add_click button is fired when the user clicks on the SubmitDetailsBtn button. This method builds a SQL execute query and passes it on to the RunSql function, which in turn executes it. The script for the add_click method is as follows:

Sub add click

Sub add_click(Source As Object, E As EventArgs)

Dim sql As string

sql = "Execute p_stock_trans @date = '"

sql = sql+ adate.text+"',@ref= '"+aref.text+"', @qty_in ="

sql = sql+ aqty_in.text+",@qty_out = "+aqty_out.text+","

sql = sql+ "@id = 'RPT', @doc_no = NULL"+", @narr= '"

sql = sql+ anarr.text+"', @code_value = "+acode_value.SelectedItem.value

RunSql(sql)

rebind()

hidePanel()

End Sub

The hidePanel function simply hides the panel (by setting its visible property to False) and sets the value property of all the TextBox controls to spaces.

The Update Mode

The DataGrid operates in the Edit mode when the edit link is clicked. The user makes the appropriate changes and clicks on the Ok link. This fires off the Grid1_Update function. The value property for all the TextBox controls is extracted, and a SQL procedure call string is built. This string is passed onto the RunSQL function, which makes the actual procedure call.

Sub Grid1_Update

Sub Grid1_Update(sender As Object, e As DataGridCommandEventArgs)

Dim sql As string

Dim vdate As String

Dim ref As String

Dim code_value As String

Dim qty_in As String

Dim qty_out As String

Dim id As String

Dim narr As String

Dim myTextBox As TextBox

'This is the key value:

'Retrieved from the DataKey, since it's a read only field

Dim doc_no As string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString

myTextBox = E.Item.FindControl("edit_date")

vdate = mytextbox.text

myTextBox = E.Item.FindControl("edit_ref")

ref = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_qty_in")

qty_in = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_qty_out")

qty_out = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_narr")

narr = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_code_value")

code_value = trim(mytextbox.text)

'Now execute stored procedure

sql = "Execute p_stock_trans @date = "

sql = sql + vdate + "',@ref= '" + ref + "', @qty_in ="

sql = sql + qty_in + ",@qty_out = "+ qty_out +" , "

sql = sql + "@id = 'STK', @doc_no = "+doc_no+", @narr= '"+narr+ "',"

sql = sql + "@code_value=" + code_value

'response.write(sql)

RunSql(sql)

rebind()

End Sub

Function RunSql

This is a generic function, which executes a SQL Action query against the database. The SQL query is passed to this function as a parameter. The Grid1_update Sub, the Add_click Sub, and the <u>Grid1_delete</u> Sub call this function to update, add, or delete a record. This function in turn calls the RunSql function of the web service and passes to it as parameters the connection string as well as the SQL action query/procedure call. If the procedure/query was executed successfully, the string "Success" is returned from the function. Otherwise, the appropriate error string is returned, which is displayed in the browser.

Sub RunSQL

Sub RunSql(vsql as string)

Dim t As New NameSpaceHersh.SQLService

Dim s As string

s = t.RunSQL(ConnStr,vSQL)

Grid1.EditItemIndex = -1

Rebind

```
if s <> "Success" then
```

Message.Text = s

Message.Style("color") = "red"

End if

End Sub

The Delete Mode

I have created a ButtonColumn having a CommandName of Delete as follows:

```
<asp:ButtonColumn Text = "Delete" CommandName = "Delete" HeaderText = "Delete"/>.
```

The OnDeleteCommand of the DataGrid fires the <u>Grid1_delete</u> function whenever the user clicks on the delete hyperlink. The <u>Grid1_delete</u> function sends a SQL delete query to the RunSql function, which deletes all tr_header and transactions records having a document number equal to the clicked doc_no.

Grid1_delete

Sub Grid1_delete(sender As Object, e As DataGridCommandEventArgs)

Dim doc_no As string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString

Dim sql As string

sql = " Delete from stock_detail where doc_no = " + cstr(doc_no)

sql = sql + " Delete from tr_header where doc_no = " + cstr(doc_no)

RunSql(sql)

rebind()

End Sub

Here is the complete code listing of StockTrans.aspx StockTrans.aspx

<%@Page Language="VB" Inherits="BaseClass" Src="StockTrans.vb" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<%@ Register TagPrefix="Hersh" TagName="nav" Src="nav.ascx" %>

<html>

<script language="VB" runat="server">

Sub Grid1_Update(sender As Object, e As DataGridCommandEventArgs)

Dim sql As string

Dim vdate As String

Dim ref As String

Dim code_value As String

Dim qty_in As String

Dim qty_out As String

Dim id As String

Dim narr As String

Dim myTextBox As TextBox

'This is the key value:

'Retrieved from the DataKey, since it's a read only field

Dim doc_no As string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString

myTextBox = E.Item.FindControl("edit_date")

vdate = mytextbox.text

myTextBox = E.Item.FindControl("edit_ref")

ref = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_qty_in")

qty_in = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_qty_out")

qty_out = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_narr")

narr = trim(mytextbox.text)

myTextBox = E.Item.FindControl("edit_code_value")

code_value = trim(mytextbox.text)

'Now execute stored procedure

sql = "Execute p_stock_trans @date = '"

```
sql = sql + vdate + "',@ref= '" + ref + "', @qty_in ="
```

```
sql = sql + qty_in + ",@qty_out = "+ qty_out +" , "
```

```
sql = sql + "@id = 'STK', @doc_no = "+doc_no+", @narr= '"+narr+ "',"
```

```
sql = sql + "@code_value=" + code_value
```

'response.write(sql)

RunSql(sql)

rebind()

End Sub

```
Sub add_click(Source As Object, E As EventArgs)
```

Dim sql As string

sql = "Execute p_stock_trans @date = "

sql = sql+ adate.text+"',@ref= '"+aref.text+"', @qty_in ="

sql = sql+ aqty_in.text+",@qty_out = "+aqty_out.text+","

```
sql = sql+ "@id = 'RPT', @doc_no = NULL"+", @narr= '"
```

```
sql = sql+ anarr.text+"', @code_value = "+acode_value.SelectedItem.value
```

RunSql(sql)

rebind()

hidePanel()

End Sub

```
Sub add_show(Source As Object, E As EventArgs)
```

AddPanel.visible = true

End Sub

```
Sub Grid1_delete(sender As Object, e As DataGridCommandEventArgs)
```

```
Dim doc_no As string = Grid1.DataKeys.Item(E.Item.ItemIndex).ToString
```

Dim sql As string

```
sql = " Delete from stock_detail where doc_no = " + cstr(doc_no)
```

sql = sql + " Delete from tr_header where doc_no = " + cstr(doc_no)

RunSql(sql)

rebind()

End Sub

</script>

<head>

<style>

a { color:black;

text-decoration:none;}

a:hover {color:red;

text-decoration:underline;}

</style>

</head>

<body style="font: 10pt verdana; background-color:ivory">

<form runat="server">

<asp:ValidationSummary runat=server

headertext="There were errors on the page:" />

<!----- Navigation Start----->

<Hersh:nav id="menu" runat = server

vGridlines = Both

vBorderColor = "Black"

vCellPadding = 7 />

```
<!----- Navigation Ends------>
```

<h3> Inventory Movements

<asp:Label id="title" runat="server"/> </h3>

<asp:Button id="Addshow" visible = "false" text="New Tranaction"

```
onclick="add_show" runat="server" />
```

<hr>

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

OnEditCommand="Grid1_Edit"

OnCancelCommand="Grid1_Cancel"

OnUpdateCommand="Grid1_Update"

OnDeleteCommand = "Grid1_delete"

DataKeyField="doc_no">

<Columns>

<asp:EditCommandColumn

EditText="Edit"

CancelText="Cancel"

UpdateText="OK"

ItemStyle-Wrap="false"

HeaderText="Edit"

HeaderStyle-Wrap="false"/>

<asp:ButtonColumn Text="Delete" CommandName="Delete"

HeaderText="Delete"/>

<asp:BoundColumn HeaderText="Doc #" ReadOnly="true"

DataField="doc_no"/>

<asp:TemplateColumn HeaderText="Ref" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("ref") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_ref" Text='<%# Container.DataItem("ref") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Date" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("date") %>'

runat="server" />

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_date" BorderStyle="None" Readonly="True"

Text='<%# Container.DataItem("date") %>'

runat="server" />

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Inventory Account" >

<ltemTemplate>

<asp:Label Text='<%# Container.DataItem("code_display") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_code_value"

Text='<%# Container.DataItem("code_value") %>'

runat="server" ReadOnly="true" BorderStyle="None" />

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Narration" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("narr") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_narr"

Text='<%# Container.DataItem("narr") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Quantity In" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("qty_in") %>'

runat="server"/>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_qty_in"

Text='<%# Container.DataItem("qty_in") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="Quantity Out" >

<ItemTemplate>

<asp:Label Text='<%# Container.DataItem("qty_out") %>'

runat="server"/>

</ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="edit_qty_out"

Text='<%# Container.DataItem("qty_out") %>'

runat="server"/>

</EditItemTemplate>

</asp:TemplateColumn>

</Columns>

<HeaderStyle BackColor="DarkRed"

ForeColor="White" Font-Bold="true"/>

<ItemStyle ForeColor="DarkSlateBlue"/>

<AlternatingItemStyle BackColor="Beige"/>

</asp:DataGrid>

<!--- insert row logic----->

<asp:Panel id="AddPanel" runat="server" Visible="false">

Add a New Transaction:

Date (Required):

<asp:TextBox id="adate" runat="server" value = "" />

<asp:RequiredFieldValidator runat=server

controltovalidate=adate

errormessage="Date is required.">*

</asp:RequiredFieldValidator>

Reference (Required/ must be unique):

<asp:TextBox id="aref" value = "" runat="server" />

<asp:RequiredFieldValidator runat=server

controltovalidate=aref

errormessage="A unique reference # is required.">*

</asp:RequiredFieldValidator>

Account Posted To:

```
<asp:DropDownList DataTextField = "code_display"
```

DataValueField = "code_value" id="acode_value"

```
runat="server" />
```

Narration:

<asp:TextBox id="anarr" value = "" runat="server" />

Quantity In:

<asp:TextBox id="aqty_in" value = 0 runat="server" />

Quantity Out:

<asp:TextBox id="aqty_out" value = 0 runat="server" />

<asp:Button id="SubmitDetailsBtn" text="Submit"

```
onclick="add_Click" runat="server" />
```

</asp:Panel>

```
<!----> Insert Logic ends
```

<hr>

<asp:Label id="Message" runat="server"/>

</form>

</body>

</html>

Here is the listing of the Code Behind file StockTrans.vb. **StockTrans.vb**

Option Strict Off

Imports System

Imports System.Collections

Imports System.Text

Imports System.Data

Imports System.Data.OleDb

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class BaseClass

Inherits System.Web.UI.Page

Protected Grid1 as DataGrid

Protected Message as label, title as label

Protected acode_value as dropdownlist, selection as dropdownlist

Protected AddPanel as Panel

Protected adate as TextBox, aref as TextBox, aqty_in as TextBox

Protected aqty_out as TextBox , anarr as TextBox

Protected addshow as button

Dim ds As New DataSet

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs) ConnStr = "Provider=SQLOLEDB; Data Source=(local); " ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;" if NOT (isPostBack) rebind End if End Sub Sub ReBind() Dim t As New NameSpaceHersh.SQLService Dim ds As DataSet 'DataSetCommand SQL = "select * from tr_header h, stock_detail s, stock_master m" SQL = SQL + " where h.doc_no = s.doc_no" SQL = SQL + " AND s.code_value = m.code_value" ds = t.Populate(ConnStr, SQL) Grid1.DataSource=ds.Tables("vTable").DefaultView Grid1.DataBind() 'populate inventory account(add mode) selection drop down list SQL = "Select * from stock_master" ds = t.Populate(ConnStr, SQL) acode_value.DataSource=ds.Tables("vTable").DefaultView acode_value.DataBind() addshow.visible = true End Sub Sub Grid1_Edit(Sender As Object, E As DataGridCommandEventArgs) Grid1.EditItemIndex = E.Item.ItemIndex ReBind() End Sub Sub Grid1_Cancel(Sender As Object, E As DataGridCommandEventArgs) Grid1.EditItemIndex = -1

```
ReBind()
End Sub
Sub RunSql(vsql as string)
 Dim t As New NameSpaceHersh.SQLService
 Dim s As string
 s = t.RunSQL(ConnStr,vSQL)
 Grid1.EditItemIndex = -1
  Rebind
 if s <> "Success" then
  Message.Text = s
  Message.Style("color") = "red"
 End if
End Sub
Sub hidePanel()
 if AddPanel.visible = true then
  AddPanel.visible = false
  'reset values
  adate.text = ""
  aref.text = ""
  aqty_in.text = ""
  aqty_out.text = ""
  anarr.text = ""
```

End if

End Sub

Chapter 26: The Inventory Balances Report

Overview

The inventory balances report displays the closing balance of all inventory items. This is the main report of an inventory management system and forms the basis for getting a valuation of the stock.

Figure 26.1 shows what it looks like.

in the second	and Market		and a second	Income and Advances	
Porta and	cory names		CLEURS D BURNING	Strenty Learner	
nventory Bal	ances				
Account.	UNITS .	Operating Bu	dance Period Clesing Balance		
Are Deep	der .	4	100		
lossed fill	Ten				
toe Tone olad	212	6	5000		
475.13	CH4	10			

Figure 26.1: The inventory balances report.

The ReBind function calls the populate function of the SQLService web service by passing it a SQL Query and connection string. A DataSet containing all the rows from the stock_master table is returned.

The ReBind Function

Sub ReBind()

Dim t As New NameSpaceHersh.SQLService

Dim ds as DataSet

'DataSetCommand

sql = "SELECT * from stock_master "

ds = t.Populate(ConnStr, SQL)

Grid1.DataSource=ds.Tables("vTable").DefaultView

Grid1.DataBind()

End Sub

The DataGrid columns property is set to display the required columns. Various templates are applied to refine the look of the DataGrid. StockBalances.aspx is the complete listing of this report. StockBalances.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<%@ Register TagPrefix="Hersh" TagName="nav" Src="nav.ascx" %>

<html>

<script language="VB" runat="server">

Dim ConnStr As String

Dim SQL As String

Sub Page_Load(Source As Object, E As EventArgs)

ConnStr = "Provider=SQLOLEDB; Data Source=(local); "

ConnStr = ConnStr+" Initial Catalog=ASPNET;User ID=sa;"

if NOT (isPostBack)

rebind

end if

End Sub

Sub ReBind()

Dim t As New NameSpaceHersh.SQLService

Dim ds As DataSet

'DataSetCommand

sql = "SELECT * from stock_master "

ds = t.Populate(ConnStr, SQL)

Grid1.DataSource=ds.Tables("vTable").DefaultView

Grid1.DataBind()

End Sub

</script>

<head>

<style>

```
a { color:black;
```

```
text-decoration:none;}
```

a:hover { color:red;

text-decoration:underline;}

</style>

</head>

<body style="font: 10pt verdana; background-color:ivory">

<!---- Navigation Start------>

<Hersh:nav id="menu" runat = server

vGridlines = Both

vBorderColor = "Black"

vCellPadding = 7 />

<!----- Navigation Ends------->

<h3>Inventory Balances </h3>

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt">

<Columns>

<asp:BoundColumn HeaderText="Account" DataField="code_display" >

<HeaderStyle Width="150px"/>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Units" DataField="uom">

<HeaderStyle Width="150px"/>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Opening Balance" DataField="opening">

<HeaderStyle Width="150px"/>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Period Closing Balance" DataField="closing">

<HeaderStyle Width="150px"/>

</asp:BoundColumn>

</Columns>

<HeaderStyle BackColor="teal" ForeColor="white" Font-Bold="true"/>

<ItemStyle ForeColor="DarkSlateBlue"/>

<AlternatingItemStyle BackColor="Beige"/>

</asp:DataGrid>

</body>

</html>

Project 3 Summary

My goal in the projects described in Projects 2 and 3 was to demonstrate practically how traditional client/server products could be revamped for the Web using SOAP, web services, and the tools provided by ASP.NET. The two "hubs" of a Management Information System are the Financial Accounting System and the Inventory Management System. I developed a Web-enabled Personal Financial System in <u>Project 1</u> and a Web-enabled Inventory Management System in this project. The Database web service developed in <u>Project 2</u> sits between both these modules and the database. In other words, both these modules talk to the database via the Database web service.

The various modules in a Management Information System can be built using any of the .NET-compliant languages. The Personal Finance Manager could be written in Visual Basic.NET and the Inventory module in C#. Both these languages provide a rich Graphical User Interface, which users have come to accept and demand. However, beneath the hood they are both Web-enabled through the Database web service. The Database web service abstracts calls and interactions to the database. Database operations like selects, inserts, and updates are implemented as simple function calls. It is relatively simple to add more modules because we are only required to write the GUI; the database functionality is already encapsulated in the Database web service.

Project 4: The GenEditAdd Control

Chapter List

- <u>Chapter 27:</u> Using the GenEditAdd Control
- <u>Chapter 28:</u> Extending the GenEditAdd Contrtol

Project 4 Overview

The GenEditAdd control can be used to insert or update database records. The ASP.NET DataGrid does not have the capability to insert records, although it does have editing capabilities. The Edit mode of the DataGrid is quite cumbersome, because you need to code a number of events in the DataGrid for the process to work. The GenEditAdd control was developed to enhance the usefulness of the DataGrid because it can be hooked up to a DataGrid to provide both editing and insertion capabilities in a consistent manner. This control works by setting various properties, and the code generation is automated. This project is actually a sequel to the GenEditAdd control developed in <u>Chapter 7</u> and that chapter is therefore essential reading before starting this project. The control developed in <u>Chapter 7</u> was kept simple so that I could explain it better. I had explained there that you could set eight properties to customize the control.

These were:

- 1. Display: This property controls the fields that get displayed.
- 2. SQL: This is the SQL query for the control.

- 3. Where: This is the where clause. If the where clause does not exist, the control presents a new entry form; otherwise, an Edit form is displayed which is pre-filled with existing data.
- 4. ConnStr: The database connection string.
- 5. KeyField: The field name of the primary field.
- 6. KeyValue: The value of the primary field.
- 7. Procedure: The stored procedure for the Insert and Update modes.
- 8. ExitPage: A link back to the calling page.

This project refines and enhances the control and also adds a number of features to it. These features are:

- 1. DropDownLists: This is a very nifty feature. You can have a field appear as a drop-down list for a user to pick values from. You can build the drop-down list from a database table (or tables) and specify display and code fields. The display field is what is displayed to the user and the code field is what gets stored in the database. For example, you could display an employee name (a char field) to the user and store an employee id (a numeric field) based on the user selection. The best part is that *it maintains state*. When a record is selected for editing, the drop-down list shows the value that was actually stored in that field. The ASP.NET DropDownList cannot do that and you have to do some programming to make that happen.
- Required Fields: GenEditAdd makes use of the ASP.NET
 <u>RequiredField</u> validation control to enforce input in required fields.
 Further it makes use of the validation summary control to give summary
 error information.
- 3. <u>Editable Fields</u>: You can tell GenEditAdd which fields are editable fields. Editable fields display textboxes to accept user input while non-editable fields display labels that cannot be edited.
- 4. <u>Field Names</u>: You can set the field name to display against the input fields. If no name is specified, the raw field name from the database is used.
- 5. InsertProcedure: This is the database-stored procedure that is called in the Insert mode.
- 6. UpdateProcedure: This is the database-stored procedure that gets called in the Update mode.

The InsertProcedure and the UpdateProcedure properties replace the single Procedure property in the <u>Chapter 7</u> implementation of GenEditAdd. This was done because I realized that you might want to use separate procedures for inserts and updates even though I personally use a single procedure for both.

Chapter 27: Using the GenEditAdd Control

This chapter gets you started with using the GenEditAdd control. I will not deal with any theory here; I leave that for subsequent chapters. I will provide instructions to enable you to use this control in your projects. The complete source code for this control can be found on the <u>Project 4</u> samples folder on the book's Web site at <u>www.premierpressbooks.com/downloads.asp</u>.

Compiling the Control

I have provided a .bat file, which will compile the GenEditAdd control to a DLL. Place this bat file along with the script file of the control (GenEditAdd.vb) in an application folder and execute it. Make sure that the bat file points to your bin folder; if not, modify the outdir variable in the bat file. Successful execution should place the GenEditAdd.dll in the bin folder of your application.

The Config File

The Config file is the link between the DataGrid and the GenEditAdd control. Here you set all the properties of the control except the property KeyValue (which is passed to it from the DataGrid) and the Where property (which is built dynamically based on the KeyValue property). Here is a sample config file:

config_master.aspx

<%@ Register TagPrefix="Hersh" Namespace="Generic" Assembly="GenEditAdd"%>

<html>

```
<script language="VB" runat="server">
```

Sub page_load(sender As Object, e As EventArgs)

if NOT (isPostBack)

Dim s As string

Dim sql As string

Dim Is_CodeValue As string

Is_CodeValue = Request.QueryString("code_value")

'- these properties discussed in chapter 7

SQL = "Select * from masters"

Gen.sql = SQL

if cint(ls_codeValue) = 0 then

Gen.Where = ""

else

```
Gen.where= " Where code_value =" + Is_CodeValue
```

end if

Gen.display = "111110"

Gen.KeyField = "code_value"

Gen.KeyValue = ls_codeValue

Gen.ExitPage = "masters.aspx"

'-New properties developed in project 3

Gen.Insertprocedure = "p_masters"

Gen.Updateprocedure = "p_masters"

```
Gen.RequiredFields = "0100000"
```

Gen.editable = "1110100"

s= "code_category;code_value;code_display;Select * from groups;"

Gen.DropDown = s

Gen.FieldNames ="id;Account Name; Group;Type; Closing Bal;Opening Bal;"

end if

End Sub

</script>

<body>

<form runat = "server" >

<Hersh:GenEditAdd id = "Gen" runat=server

ConnStr = "Provider=SQLOLEDB; Data Source=(local);

```
Initial Catalog=ASPNET;User ID=sa;" />
```

</form>

</body>

</html>

Note that the code_value variable (which is actually the KeyValue property) is passed to this page from the calling page that hosts the DataGrid. The DataGrid will pass a valid primary key or a value of zero to the config page. A valid primary key tells GenEditAdd that the Edit mode is expected and the Where property is dynamically built. A primary key value of zero tells GenEditAdd that the user wants to add a new record and the Where property of the control is set to a blank string.

Hooking GenEditAdd to a DataGrid

The DataGrid requires two hyperlink columns, one for the Add mode and the other for the Edit mode. These hyperlinks navigate to the config file and pass on a code_value of 0 in case of the Add mode or a valid primary key in case of the Edit mode. This is shown in the following extract from the web form masters.aspx (I will deal with this form later in this chapter):

The GenEditAdd hyperlinks in masters.aspx

<property name="Columns">

<asp:HyperLinkColumn Text="Edit" DataNavigateUrlField="code_value"

DataNavigateUrlFormatString="config_masters.aspx?code_value={0}"/>

<asp:HyperLinkColumn Text="Add" DataNavigateUrlField="code_value"

DataNavigateUrlFormatString="config_masters.aspx?code_value=0"/>

.....

.....

</property>

And that's all you need to do to get the GenEditAdd control up and running. <u>Table 27.1</u> provides a quick reference guide to the GenEditAdd control.

Table 27.1: GenEditAdd Quick Reference

Property	Description
SQL	The SQL query (without the where clause).
	<pre>Example: Gen.SQL = "Select * from Masters"</pre>
Where	The where clause. This is also the toggle between the Edit and Add modes; if a where clause exists, GenEditAdd displays the Edit mode, else it displays the Add mode.
	<pre>Example: ls_CodeValue = Request.QueryString("code_value") If cint(ls_codeValue) = 0 Then Gen.Where = "" Else Gen.where= " Where code_value =" + ls_CodeValue End If</pre>
Display	This is a string of 0s and 1s. A 0 means don't show a field and a 1 means show it. For example, the setting 011 will hide the first field and display the next two fields.
KeyField	The primary key field name.
KeyValue	The primary key value.
Exit Page	The URL of the web form to exit (return) to from the config web form. Typically this web form will be the web form on which the DataGrid resides. When you click on the Add or Edit links on the web form that hosts the DataGrid, you are directed to the config web form, on which the GenEditAdd control resides. After finishing adding or modifying a record, you would want to return back to the DataGrid web form. The ExitPage property builds a hyperlink on the top of the config web form. Clicking on that allows you to navigate back to the calling web form.
ConnStr	The connection string. For example: ConnStr = "Provider=SQLOLEDB; Data Source=(local); Initial _ Catalog=ASPNET;User ID=sa;"
DropDown	Builds any number of DropDown list columns. Each

able 27.1: GenEditAdd Quick Reference						
Property	Description					
	 DropDown requires you to specify four properties. Each property is specified as a string, and a semicolon separates each property. Thus if you want to create two DropDowns, you would have eight properties (four for each DropDown), each property separated by a semicolon. The four properties are as follows (the order is important): The name of the field that should have a DropDown. The code field: This is the field of the DropDown that gets stored in the database; for example, a numeric employee code. The display field: This is the DropDown list field that is displayed to the user; for example, a descriptive employee name. The SQL statement. This is the SQL clause that populates the DropDown list. It should include a display and a code column (that is, it should fetch at least two columns unless the display and code columns are the same). 					
	Example: Dim s As string 'This is dropdown 1 s=					
	<pre>"code_category;code_value;code_display;Sele ct * _ from groups;" 'This is dropdown 2 s=s + "code_display;code_display;code_display;Sel oct *</pre>					
	from masters;" Gen.DropDown = s					
	This will create two DropDown lists as follows: DropDown List 1: is attached to the code_category field. The code field is the code_value field of the groups' table and the display field is the code_display field of the groups' table. The SQL statement is Select * from groups. I am displaying a DropDown list to help users select groups. The user sees the descriptive group name (code_display from groups table) but the numeric code (code_value) is stored in the code_category field.					
	DropDown List 2: This list is attached to the code_display column. I want to display what I want to save. In other words the display and code fields are same. This DropDown list displays all records from the masters table.					
<u>Required Field</u>	This is a string of zeros and ones. A zero implies that a field is not a required field, whereas a one means that it is. The user must make an entry in a required field. If this field is left blank, an error message is displayed. The errors will also be displayed in a ValidationSummary control. The record will not be saved unless all required fields are filled.					
able 27.1: GenEditAdd Quick Reference						
---------------------------------------	---	--	--	--	--	--
Property	Description					
	Example: Gen.RequiredFields = "0100000"					
	The second field is a required field in the above example.					
Editable Field	This is also a string of zeros and ones. A one means that the field can be edited, and a zero means that a field can only be displayed and cannot be edited by the user.					
	Example: Gen.editable = "1110100"					
	The first, second, third, and fifth fields can be edited by the user in this example.					
<u>Field Names</u>	This is a string that specifies the field names to be displayed against columns of the form. Semicolons separate the field names. If this property is not specified, the raw database column names are used as the column labels. For example: Gen.FieldNames ="id;Account Name; Group;Type; Closing Bal:Opening Bal:"					
InsertProcedure	The database-stored procedure that is called in the Insert mode.					
UpdateProcedure	The database-stored procedure that is called in the Update mode.					

An Example

I will hook up the GenEditAdd control to a DataGrid and use it to insert and update records to the masters table. The config file (config_masters.aspx) was listed earlier in this chapter .The code extract required to set up the DataGrid in the calling web form (masters.aspx) is as follows:

Extract from Masters.aspx hooking a DataGrid to GenEditAdd

<asp:DataGrid id="Grid1" runat="server"

AutoGenerateColumns="false"

BackColor="White"

BorderWidth="1px" BorderStyle="Solid" BorderColor="Tan"

CellPadding="2" CellSpacing="0"

Font-Name="Verdana" Font-Size="8pt"

OnDeleteCommand = "Grid1_delete"

DataKeyField="code_value">

<Columns>

<asp:HyperLinkColumn Text="Edit" DataNavigateUrlField="code_value"

DataNavigateUrlFormatString="config_masters.aspx?code_value={0}"/> <asp:HyperLinkColumn Text="Add" DataNavigateUrlField="code_value" DataNavigateUrlFormatString="config_masters.aspx?code_value=0"/> <asp:ButtonColumn Text="Delete" CommandName="Delete" HeaderText="Delete"/> <asp:BoundColumn HeaderText="Account" DataField="code_display"> <HeaderStyle Width="150px"></HeaderStyle> </asp:BoundColumn HeaderText="Group" DataField="category">

<HeaderStyle Width="150px"></HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Type" DataField="type">

<HeaderStyle Width="150px"></HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Opening" DataField="opening">

<HeaderStyle Width="150px"></HeaderStyle>

</asp:BoundColumn>

<asp:BoundColumn HeaderText="Closing" DataField="closing">

<HeaderStyle Width="50px"></HeaderStyle>

</asp:BoundColumn>

</Columns>

<HeaderStyle BackColor="DarkRed" ForeColor="White" Font-Bold="true" >

</HeaderStyle>

<ItemStyle ForeColor="DarkSlateBlue">

</ltemStyle>

<AlternatingItemstyle BackColor="Beige">

</AlternatingItemstyle>

</asp:DataGrid>

Figure 27.1 shows what the masters.aspx looks like.

8	0	6 6	÷	C	6.	5	1		f. 90
105 07	NUL Stop Fielend	h Home Searc	t favoites	Holoy	Mal	Piex	10	Meconiper	-
									1
hart of Acces	ants:								
Delet	Account	Group	Туре	Open	ing Cleang	I			
Edit Add Celets	Visa Card	Correct liabilities	2	0	0				
Loit Add Celets	Wells Farge Checking	Cesh is hand	A		0				
Edit Add Celeta	Vise Card	Cesh is herd	A		0				
Edit Add Delete	Salary	Sales account	1						
Edit Add Celete	Interest Income	Sales account	1		0				
Edit Add Celeta	Rast	Furchase account	1		0				
Edit Add Celeta	USINER	Furchase account	E		. 0				
Edit Add Celets	Medical Expension	Furchase account.	E		. 9				
Ecit Add Celets	test 9	Capital account	A		10				
Edit Add Delets	Wells Fargo Chacking	Fixed accets	A		0				
Ecit Add Colets	Enterest Income	Rood assets	A		31				
Edit Add Celets	Salary	Revenue accounts	н		1.0				
Edit Add Celets	Salary	Eranch/divisions	A		90				
Edit Add Celete	Rest	Investments	A		2				
Edit 3.64 Celeta	Galary	Capital account	L.		2				

Figure 27.1: Masters.aspx with edit, add, and delete links.

When you click on the Edit link, the GenEditAdd displays in the Edit mode, as shown in



Figure 27.2: GenEditAdd in the Edit mode allows modification of records.

Note that GenEditAdd populates itself with the record details from the database. The DropDown list is populated from the groups table and scrolls down to the correct group associated with the record. Clicking on the Back URL will take you back to the calling page.

When you click on the Add link on the DataGrid, GenEditAdd is activated in the Insert mode, enabling you to add new records. (See Figure 27.3.)

Tes Tox	Too Mose	at Tose	Det								_	
gidens (P) i	#p.//127.00.1/	WSPEcok/w	/p/GerE.3kA	dd/config_m	eden.expe?	code_value=0	1				-	50
Ge .		500	Petroh	Hone	Search	Favoites	C Hatoy	Mai	BPHK	E.	Meconger	
<u>n-k</u>												
dd a Nev	v Record:											
covers, Na	Castala	one int	-	2								
lucing Bal	Capital a	CC04M	- 1									
second in the												
ual es	in the second											
dd Ca	icel											
dd Ca	icel											
dd Ca	icel											
dd Ca	scel											
dd Ca	cel											
dd Ca	icel											
dd Ca	icel											
dd Ca	cet											
dd Car	icel											
dd Ca	icel											
dd Ca	icel											
dd Ca	icel											
4dd Ca	icel											

Figure 27.3: The Add mode of GenEditAdd allows insertion of new records.

Chapter 28: Extending the GenEditAdd Control

The foundation of the GenEditAdd control was laid in <u>Chapter 7</u>, "<u>Custom Controls.</u>" This project develops the control further and incorporates functionality like drop-down lists, required field validation, specifying read-only fields, and providing descriptive field names to input columns. <u>Chapter 7</u> should be read before starting this project.

Drop-down List Columns

The GenEditAdd control has the capability of rendering any column as a drop-down list column. The control must know four things for each drop-down list column that it must render:

- The DDLColumn. This is the name of the database field that has to be rendered as a drop-down list column on the form.
- The CodeField. This is the field whose value will be stored in the database.
- The DisplayField. This is the field that the user will see.
- The DDLSQL. This is the SQL query that will select the records for the dropdown list.

I will call these four properties "sub-properties" of the main property DropDown. The property DropDown is specified by the user as a string, with each of the above sub-properties separated by a semicolon. The syntax for the property DropDown is: DropDown = DDLColumn; CodeField; DisplayField; DDLSQL. These sub-properties should be specified in exactly this order.

If you want to create two drop-downs, you would have eight sub-properties (four for each), separated by a semicolon. For example:

Dim s As string

'First drop-down list

s= "code_category;code_value;code_display;Select * from groups;"

'second drop-down list

s=s + "code_display;code_display;Select * from masters;"

Gen.DropDown = s

<u>Table 28.1</u> lists the four sub-properties of each drop-down list column passed to the GenEditControl's property DropDown in the above example.

Table 28.1: The Four Drop-Down List Sub-properties

Index	DDLColumn	Code Field	Display Field	DDLSQL
0	code_category	code_value	code_display	Select * from groups
1	code_display	code_display	code_display	Select * from masters

In the above example above, the two drop-down lists are used on a data input form for the masters table. The first list is used to fill in the masters code_category field (the user is shown the names of groups from the groups table and its code_value is stored upon user selection) and the second to select the descriptive name of a master record. This code would reside in a config file as explained in <u>Chapter 27</u>. Gen is the id given to the GenEditAdd control in that file.

What we pass to the control is a really long string that contains the property values separated by semicolons. The ParseDropDown method (which gets called in the CreateChildControls method) parses the string and stores the values in four arrays: DDLColumn, CodeField, DisplayField, and DDLSQL. All the values pertaining to the same drop-down list column will be stored with the same index number. For example, property values for the first drop-down list column will be stored as:

DDLColumn(0) = "code_category"

CodeField(0) = "code_value"

DisplayField(0) = "code_display"

DDLSQL(0) = "Select * from groups"

Similarly property values for the second drop-down list column will be stored as follows:

DDLColumn(1) = "code_display"

CodeField(1) ="code_display"

DisplayField(1) = "code_display"

DDLSQL(1) = "Select * from masters"

A function called GetDDLIndex returns the index associated with a drop-down list column. This function accepts the drop-down list column name as an input parameter and returns the index assigned to it as follows:

Function GetDDLIndex(vDDLColumn as string) as integer

'Pass in the column where a drop-down list must appear

' this function returns the index where its details are

' stored in the array

Dim i As integer, vKey As integer

For i= 0 to UBound(DDIColumn)

```
If DDIColumn(i) = vDDLColumn
```

vKey = i

End If

Next

Return vKey

End Function

When we know this index, we can extract the other three sub-properties associated with the drop-down list column. I have used the following code in many places within the GenEditAdd control to extract the sub-properties using the assigned index value.

Dim idx As integer

Dim vDisplayField As string

Dim vCodeField As string

Dim vDDLSql As string

Dim VFieldName As string

If isDropDown(c.ToString) Then

'get the index & other parameters

idx = GetDDLindex((c.ToString))

vDisplayField =DisplayField(idx)

vCodeField =CodeField(idx)

vDDLSql =DDLSql(idx)

End If

The isDropDown function is used to determine whether a given column is a drop-down list column or not. This function simply loops the DDLColumn array and checks if the column name passed to it as a parameter is included in this array. If it is, it means that the column is a drop-down list column and the Boolean true is returned.

Function isDropDown(vDDLColumn as string) as boolean

```
Dim i As integer, vflag As integer
```

vFlag = -1

For i= 0 to UBound(DDIColumn)

If DDIColumn(i) = vDDLColumn

vflag = 1

End if

Next

If vFlag = 1 Then

Return True

Else

Return False

End If

End Function

The function ParseDropDown() is responsible for parsing the GenEditAdd property DropDown and storing it into the four sub-property arrays DDLColumn, CodeField, DisplayField, and DDLSql. It makes use of the Visual Basic split function. The split function takes two arguments—the string to parse and a delimiter. The delimiter in this case is the semicolon. I want to extract each word separated by a semicolon and the split function does just that. It puts the parts of the split string in the object (array) strChar. It is now a simple task to loop this object. In the inner loop, I have a counter variable called count that gets incremented on each pass. In this inner loop, I will loop the object strChar four times to get to the four sub-properties. On the first pass the count variable equals one and the first element of the strChar object is identified as the DDLColumn sub-property, similarly the second element is identified as the DDLSQL subproperty. On count number four, I reset the count variable to zero, so that I can start processing the next drop-down list column (remember each drop-down list column has four sub-properties, and thus after a count of four, I must start over). All four subproperties are stored in their respective arrays with the *same* index (idx) number. Thus the four arrays are related to each other by their index numbers. I now increment the idx number by one and the outer loop starts to work on the next set of four sub- properties.

```
Function ParseDropDown()
 Dim strChar As Object
 Dim s As String
 Dim j As Integer
 Dim count As Integer
 Dim idx As integer
 count = 1
 idx = 0
 strChar = Split(DropDown, ";")
 For j = 0 To UBound(strChar)
  If Len(strChar(j)) = 0 Then
  Else
   s = CStr(strChar(j))
   If count = 1 Then
     DDLColumn(idx) = s
   Elseif count = 2 Then
     CodeField(idx) = s
   Elseif count = 3 Then
     DisplayField(idx) = s
   Elseif count = 4 Then
     DDLSql(idx) = s
     count = 0
     idx = idx + 1
   End If
   count = count + 1
  End If
 Next
End Function
```

To help you understand these functions, I have created a web form DropDown_explain.aspx. This file is not part of the GenEditAdd control. Its sole purpose is to explain the code used to parse the DropDown property and show how the property values are loaded into arrays. The GenEditAdd control has quite a bit of code. This listing would help in understanding the code for building the drop-down lists as it isolates the drop-down list functions from the rest of the code.

DropDown_explain.aspx

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.OleDb" %>

<%@ Import Namespace="System.Web" %>

<%@ Import Namespace="System.Web.UI" %>

<%@ Import Namespace="System.Web.UI.WebControls" %>

<html>

<script language="VB" runat="server"> Public DDLColumn(10) As string Public CodeField(10) As string Public DisplayField(10) As string Public DDLSql(10) As string Public s As String Sub page_load(sender As Object, e As EventArgs) ParseDropDown End Sub Sub set1_Click(Sender As Object, E As EventArgs) Dim i As integer i = GetDDLindex("1") s= "DDLColumn = " + DDLColumn(i) s = s + " CodeField=" +CodeField(i) s = s + " DisplayField=" +DisplayField(i) s = s + " DDLSQL=" +DDLSQL(i) response.write(s) End Sub Sub set2_Click(Sender As Object, E As EventArgs) Dim i As integer i = GetDDLindex("a") s= "DDLColumn = " +DDLColumn(i) s = s + " CodeField=" +CodeField(i) s = s + " DisplayField=" +DisplayField(i) s = s + " DDLSQL=" +DDLSQL(i) response.write(s) End Sub

Sub set3_Click(Sender As Object, E As EventArgs)

Dim i As integer

```
i = GetDDLindex("p")
```

s= "DDLColumn = " +DDLColumn(i)

```
s = s + " CodeField=" +CodeField(i)
```

```
s = s + " DisplayField=" +DisplayField(i)
```

```
s = s + " DDLSQL=" +DDLSQL(i)
```

```
response.write(s)
```

End Sub

```
Function GetDDLIndex(vDDLColumn as string) as integer
```

'Pass in the column where a drop down list must apper

' this function returns the row where its details are

' stored in the array

Dim i As integer, vKey As integer

For i= 0 to UBound(DDIColumn)

If DDIColumn(i) = vDDLColumn

vKey = i

End if

Next

Return vKey

End Function

Function ParseDropDown()

Dim strChar As Object

Dim s As String

Dim j As Integer

Dim count As Integer

Dim idx As integer

count = 1

idx = 0

strChar = Split("1;2;3;4;a;b;c;d;p;q;r;s;", ";")

```
For j = 0 To UBound(strChar)
   If Len(strChar(j)) = 0 Then
   Else
    s = CStr(strChar(j))
     If count = 1 then
      DDLColumn(idx) = s
     Elseif count = 2 then
      CodeField(idx) = s
     Elseif count = 3 then
      DisplayField(idx) = s
     Elseif count = 4 then
      DDLSql(idx) = s
      count = 0
      idx = idx + 1
    End if
    count = count + 1
   End If
  Next
 End Function
</script>
<body>
 <form runat = "server" >
  <asp:button text="Set1(1;2;3;4)" Onclick="set1_Click" runat=server/>
  <asp:button text="Set2(a;b;c;d)" Onclick="set2_Click" runat=server/>
  <asp:button text="Set3(p;q;r;s)" Onclick="set3_Click" runat=server/>
 </form>
```

</body>

</html>

The ParseDropDown function parses the string 1;2;3;4;a;b;c;d;p;q;r;s; and splits it into three sets. Each set has four properties. Thus, set #1 = 1,2,3,4; set #2 = a,b,c,d; and set #3 = p,q,r,s. The four sub-properties in each set are the DDLColumn, the CodeField, the DisplayField, and the DDLSQL. There are three buttons (having captions of Set1, Set2, and Set3) on the form, which display the member properties of each set when clicked.

Now let's go back to the GenEditAdd control and look at code for the drop-down list columns. I will first discuss the rendering of drop-down list columns in the Update mode of GenEditAdd. The Update mode is when the GenEditAdd controls allow modification of an existing record. In the source file (GenEditAdd.vb) this section is marked as the "UPDATE MODE" and is part of the function CreateChildControls.

Drop-down List Columns in the Update Mode of GenEditadd

!*********

'UPDATE MODE

'-----Drop Down List------

For Each r in t.Rows

For Each c in t.Columns

'Get the field name

vFieldName = FieldNamesArray(FieldsCount)

if len(vFieldName) < 1 Then

vFieldName = c.ToString

End If

.....

.....

.....

If isDropDown(c.ToString) Then

'get the index & other parameters

idx = GetDDLindex((c.ToString))

vDisplayField =DisplayField(idx)

vCodeField =CodeField(idx)

vDDLSql =DDLSql(idx)

me.Controls.Add(new LiteralControl(""))

Dim DDL As New DropDownList

```
DDL.ID = c.ToString
 DDL.DataTextField = vDisplayField
 DDL.DataValueField = vCodeField
 me.Controls.Add(DDL)
 '-----Populate the drop down------
 Dim mSql As String
 msql = vDDLSQL
 myConnection = New OleDbConnection(ConnStr)
 myCommand = New OleDbDataAdapter(mSql, myConnection)
 myCommand.Fill(ds, c.ToString)
 DDL.DataSource = ds.Tables(c.ToString).DefaultView
 DDL.DataBind
 'set the display field
 Dim dv2 As dataview
 dv2 = new DataView(ds.Tables(c.ToString))
 If c.DataType.ToString = "System.String" Then
  msql = vCodeField + " = "" + r(c).ToString +""
 Else
  msql = vCodeField + " = " + r(c).ToString
 End If
 dv2.RowFilter = msql
 DDL.Selecteditem.text = dv2(0)(vDisplayField).ToString
 DDL.Selecteditem.value = dv2(0)(vCodeField).ToString
 me.Controls.Add(new LiteralControl(""))
Else
'-----Text Boxes------
End If
.....
.....
.....
```

FieldsCount = FieldsCount + 1

Next c

Next r

The code loops through the columns collection of the DataTable. For each column, it checks if the column is a drop-down list column by calling the function isDropDown. If it is, then it calls the function GetDDLIndex to get the index associated with that column. Equipped with this index, it extracts the other three sub-properties associated with this drop-down list column list and stores them in the variables vDisplayField, vCodeField, and vDDLSql respectively. It then adds an ASP.NET DropDownList control and sets its DataTextField property equal to the value stored in the variable vDisplayField and its DataValueField property equal to the value stored in the variable vCodeField . It uses the SQL query stored in the variable vDDLSql to populate an OleDbDataAdapter, which in turn populates a DataSet. The default view of this DataSet is used to bind the DropDownList control.

The DropDownList control scrolls to the correct display field in the drop-down list. The ASP.NET DropDownList control does not have built-in support for this type of functionality. The GenEditAdd control achieves this functionality by applying a filter to locate the current value of the field in the list and setting the SelectedItem.Text and the SelectedItem.Value of the ASP.NET DropDownList control to the filtered values as follows:

'set the display field

Dim dv2 As dataview

dv2 = new DataView(ds.Tables(c.ToString))

If c.DataType.ToString = "System.String" Then

msql = vCodeField + " = '" + r(c).ToString +"'"

Else

msql = vCodeField + " = " + r(c).ToString

End If

dv2.RowFilter = msql

DDL.Selecteditem.text = dv2(0)(vDisplayField).ToString

DDL.Selecteditem.value = dv2(0)(vCodeField).ToString

The code for the drop-down list column functionality in the Insert mode is similar to the Update mode except that in this mode the ASP.NET DropDownList control does not have to maintain the state; hence, the state maintenance logic is not required.

When the user is finished adding or modifying the form, he clicks on the button called AddButton. This fires the function AddBtn_Click. This function makes a call to the isDropDown function. If this function returns true then the user input is extracted using the SelectedItem.value property of the ASP.NET DropDownList control. Otherwise the user input is extracted by using the text property of the ASP.NET TextBox control. The logic for extracting user input is same in both the insert and update modes, and is as follows:

If isDropDown(c.ToString) Then

Dim vdropdown As DropDownList

vdropdown = me.FindControl(c.ToString)

```
column = c.ToString
value = vdropdown.SelectedItem.value
Else
Dim tb As TextBox
tb = me.FindControl(c.ToString)
column = c.ToString
Value = tb.text
End If
```

Required Field

A required field is a field that must be filled in by the user. If left unfilled, the record will not save and an error message will be displayed dynamically when the user tabs out of the required field. In addition, all the error messages will be consolidated and shown as a summary.

The required field functionality is implemented using the ASP.NET RequiredFieldValidator control and the ASP.NET ValidationSummary controls. A property called RequiredFields acts as the interface between the user and GenEditAdd. The user provides a string of zeros and ones to this property as follows:

Gen.RequiredFields = "0100000"

A 1 implies that the field represented by the position of the 1 is a required field. In the above example, the second field is a required field.

The RequiredFields property has a set and a get function as usual:

Public Property RequiredFields as string

```
Get
Return State("Is_RequiredFields")
End Get
Set
State("Is_RequiredFields") = value
End Set
```

End Property

The property value is padded with a number of zeros (just in case the user forgets to set this property) and assigned to a string variable as follows:

In both the Update and Insert modes of CreateChildControls, the following code snippet creates and associates an ASP.NET RequiredFieldValidator with the field:

'----- Required field ------

If vRequired.chars(FieldsCount) = "1"

Dim vReq As New RequiredFieldValidator

vReq.controltovalidate = c.ToString

vReq.errormessage= "Please enter " + c.ToString

me.Controls.Add(vReq)

End If

FieldsCount is a variable that is incremented within the loop. This is the *serial number* of the field; that is, the cardinal position of the field in the field's collection. The chars method of the String class returns the value of a string at a certain position. This method is used in the code snippet vRequired.chars(fieldcount) to check if the field with the serial number represented by the FieldsCount variable has a value of 1. If it has, it means that the field is a required field and an ASP.NET RequiredFieldValidator control must be attached to it. The name of the database field which must be validated is provided by the c.ToString variable.

An ASP.NET ValidationSummary control is also added as follows:

'-----Validation Summary

me.Controls.Add(new LiteralControl("
>"))

Dim vSummary As New ValidationSummary

vSummary.headertext="There were errors on the page:"

me.Controls.Add(vSummary)

That is all the code required for the Required Field functionality.

Editable Fields

You can tell GenEditAdd which fields are editable and which are not. Editable fields display ASP.NET TextBox controls to allow user input whereas non-editable fields just display the field value as labels and a user cannot change the displayed value.

This interface to this functionality is via the property Editable as follows:

```
Public Property editable as string
```

```
Get
Return State("Is_editable")
End Get
Set
State("Is_editable") = value
End Set
```

End Property

The user sets this property to a string of zeros and ones—a zero means that the field represented by the position of the zero is non-editable and a one means that it is editable. For example, the property setting Gen.editable = "1110100" implies that the first three and the fifth fields are editable (and the rest are not).

In the Update mode of the CreateChildControls method, the following script checks for this property:

'-----Read only field hence label

If veditable.chars(FieldsCount) = "0"

me.Controls.Add(new LiteralControl(""))

me.Controls.Add(new LiteralControl(r(c).ToString))

me.Controls.Add(new LiteralControl(""))

'-----Editable Fields------

Else

'DropDown Field or Text Box

End If

The method chars(FieldsCount) extracts the value from the string stored in the veditable variable at the cardinal position specified by the variable FieldsCount. As explained earlier, the FieldsCount variable is incremented in the loop and can be thought of as the serial number of the field. If the value so extracted equals zero, then the field is non-editable and a label is displayed which shows the stored value of the field. Otherwise, the field is editable and an ASP.NET TextBox control or an ASP.NET DropDownList control is created. The drop-down list column creation was discussed earlier in this chapter.

In the Insert mode of CreateChildControls, a non-editable field should never be displayed; thus, it is lumped together with the non-displayable columns and the key-field column. These three types of fields should never be displayed and the following code achieves this:

'Don't show this field

If vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField _ or

veditable.chars(FieldsCount) = "0" Then

Else

'Editable fields

End If

This logic is replicated in the event handler AddBtn_Click. This event handler gets called when the AddButton is clicked. The code here loops through all the TextBox controls in the input form, extracts their value property, and builds a stored procedure call string. It does not need to deal with three types of columns (that is, the display-only, the non-editable, and the key-field columns) because the user cannot change their values. The logic above excludes these fields from the procedure call string.

Field Names

The FieldNames property enables you to provide custom captions for the various TextBox controls residing on the input form. If this property is not set, the raw database field names are used as caption for the TextBox controls. Here is an example of setting this property:

Gen.FieldNames ="id;Account Name; Group;Type; Closing Bal;Opening Bal;"

As you can note, this property contains the custom captions, separated by semicolons. In the preceding example, the first TextBox control will be captioned id, the second Account Name, and so on. In a manner similar to the ParseDropDown function (which I discussed in relation to building the drop-down list columns), the ParseFieldNames function parses the FieldName property and stores the constituent field names in an array called FieldNamesArray using the Visual Basic Split function.

Function ParseFieldNames()

This function parses the FieldNames property Dim strChar As Object Dim s As String Dim j As Integer Dim count As Integer If len(FieldNames) <1 Then Exit function

```
End If

strChar = Split(FieldNames, ";")

For j = 0 To UBound(strChar)

If Len(strChar(j)) = 0 Then

Else

s = CStr(strChar(j))

FieldNamesArray(j) = s

End If

Next

End Function
```

Once this is done, it is a simple matter to extract this name in the main loop of the CreateChildControls function as follows:

```
For Each r in t.Rows
```

For Each c in t.Columns 'Get the field name vFieldName = FieldNamesArray(FieldsCount) If len(vFieldName) < 1 Then

vFieldName = c.ToString

End If

FieldsCount = FieldsCount + 1

Next c

Next r

The code retrieves the field name from the FieldNamesArray based on the Fields- Count variable, which is the serial number of the field. If the length of the name is greater than one (that is, a name exists), the user-defined name is used; else, the raw database field name is used as the TextBox control caption.

Stored Procedure Names

GenEditAdd enables you to specify the stored procedures to use in the Insert and the Update modes. This is done through two property settings: the InsertProcedure and the UpdateProcedure, as follows:

Public Property InsertProcedure as string

```
Get
Return State("Is_Insertprocedure")
End Get
Set
State("Is_Insertprocedure") = value
End Set
End Property
Public Property UpdateProcedure as string
Get
```

```
Return State("Is_Insertprocedure")
End Get
Set
State("Is_Insertprocedure") = value
End Set
End Property
```

In the config file, you would set these properties as follows:

Gen.Insertprocedure = "p_Insertmasters"

Gen.Updateprocedure = "p_Updatemasters"

The event handler AddBtn_Click when building the stored procedure call string uses this property setting as follows:

If mode = "update" Then

s = "Execute " + Updateprocedure + ""

s = s + etc etc...

Else

s = "Execute " + Insertprocedure + ""

s = s + etc etc...

End If

The Complete Code Listing

The following is the complete code listing of the GenEditAdd control:

Option Strict Off Imports System Imports System.Web

GenEditAdd.vb

Imports System.Web.UI

Imports System.Web.UI.WebControls

Imports System.Data

Imports System.Data.OleDb

Imports Microsoft.VisualBasic

Imports System.Collections

Namespace Generic

Public Class GenEditAdd : Inherits Control : Implements INamingContainer

Private Is_display as string

Private Is_where as string

Private Is_sql as string

Private Is_ConnStr as string

Private Is_keyField as string

Private Is_keyValue as string

Private Is_Insertprocedure as string

Private Is_Updateprocedure as string

Private Is_exitpage as string

Private It_datatable as datatable

Private Is_mode as string

Private Is_RequiredFields as string

Private Is_editable as string

Private Is_DropDown as string

Private Is_fieldNames as string

Protected mytbl as table

Protected DDLColumn(10) as string

Protected CodeField(10) as string

Protected DisplayField(10) as string

Protected DDLSql(10) as string

Protected FieldNamesArray(15) as string

Public Property FieldNames as string

Get

Return Cstr(ViewState("ls_FieldNames"))

End Get

Set

ViewState("Is_FieldNames") = value

End Set

End Property Public Property InsertProcedure as string Get Return Cstr(ViewState("ls_Insertprocedure")) End Get Set ViewState("Is_Insertprocedure") = value End Set End Property Public Property UpdateProcedure as string Get Return Cstr(ViewState("ls_Insertprocedure")) End Get Set ViewState("Is_Insertprocedure") = value End Set End Property Public Property DropDown as string Get Return Cstr(ViewState("Is_DropDown")) End Get Set ViewState("ls_DropDown") = value End Set End Property Public Property editable as string Get Return Cstr(ViewState("ls_editable"))

Set

End Get

ViewState("ls_editable") = value End Set End Property Public Property RequiredFields as string Get Return Cstr(ViewState("Is_RequiredFields")) End Get Set ViewState("ls_RequiredFields") = value End Set End Property Public Property Mode as string Get Return Cstr(ViewState("ls_mode")) End Get Set ViewState("Is_mode") = value End Set End Property Public Property ExitPage as string Get Return Cstr(ViewState("ls_exitpage")) End Get Set ViewState("Is_exitpage") = value End Set End Property Public Property t as datatable Get Return It_datatable

End Get

Set

lt_datatable = value

End Set

End Property

Public Property KeyField as string

Get

Return Cstr(ViewState("ls_keyfield"))

End Get

Set

ViewState("ls_keyfield") = value

End Set

End Property

Public Property KeyValue as string

Get

Return Cstr(ViewState("ls_keyvalue"))

End Get

Set

ViewState("ls_keyvalue") = value

End Set

End Property

Public Property display as string

Get

Return Cstr(ViewState("ls_display"))

End Get

Set

ViewState("Is_display") = value

End Set

End Property

Public Property Where as string

Get Return Cstr(ViewState("ls_where")) End Get Set ViewState("Is_where") = value End Set End Property Public Property SQL as string Get Return Cstr(ViewState("ls_sql")) End Get Set ViewState("ls_sql") = value End Set End Property Public Property ConnStr as string Get Return Cstr(ViewState("Is_ConnStr")) End Get Set ViewState("ls_ConnStr") = value End Set End Property Protected Overrides Sub CreateChildControls() '-----Parse the DropDown property ParseDropDown '---Parse FieldNames ParseFieldNames '-----Input Form------Dim dv As DataView

```
Dim myConnection As OleDbConnection
Dim myCommand As OleDbDataAdapter
Dim ds As New DataSet
Dim vSq As string
If len(Where) < 1 then
 vSql = SQL
 mode = "insert"
Else
 vSql = SQL + Where
 mode = "update"
End If
myConnection = New OleDbConnection(ConnStr)
myCommand = New OleDbDataAdapter(vSql, myConnection)
myCommand.Fill(ds, "vtable")
dv = new DataView(ds.Tables("vtable"))
Dim Fields As Integer
t = dv.Table
Dim r As DataRow
Dim c As DataColumn
Dim cell As TableCell
Dim row As DataRow
Dim Fieldscount As integer
Dim s As string
Dim vdisplay As string
Dim vRequired As string
Dim veditable As string
Dim idx As integer
Dim vDisplayField As string
Dim vCodeField As string
Dim vDDLSql As string
```

Dim VFieldName As string

FieldsCount = 0

s = "Back"

me.Controls.Add(new LiteralControl(s))

s= ""

me.Controls.Add(new LiteralControl(s))

me.Controls.Add(new LiteralControl(""))

If mode = "insert" then

s="<td colspan='2' bgcolor='#aaaadd'"

s= s + " style='font:10pt verdana'>Add a New Record:

me.Controls.Add(new LiteralControl(s))

Else

```
s="<td colspan='2' bgcolor='#aaaadd' "
```

s= s + " style='font:10pt verdana'>Edit Record: "

me.Controls.Add(new LiteralControl(s))

End If

me.Controls.Add(new LiteralControl(""))

UPDATE MODE

If mode = "update" then

For Each r in t.Rows

For Each c in t.Columns

'Get the field name

vFieldName = FieldNamesArray(FieldsCount)

If len(vFieldName) < 1 then

vFieldName = c.ToString

End If

'Don't show this field

If vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField then

Else

me.Controls.Add(new LiteralControl(""))

'label

me.Controls.Add(new LiteralControl(""))

me.Controls.Add(new LiteralControl(vFieldName))

me.Controls.Add(new LiteralControl(""))

'value

'-----Read only field hence label

If veditable.chars(FieldsCount) = "0"

me.Controls.Add(new LiteralControl(""))

me.Controls.Add(new LiteralControl(r(c).ToString))

me.Controls.Add(new LiteralControl(""))

'-----Editable Fields------

Else

'-----Drop Down List-----

If isDropDown(c.ToString) then

'get the index & other parameters

idx = GetDDLindex((c.ToString))

vDisplayField =DisplayField(idx)

vCodeField =CodeField(idx)

vDDLSql =DDLSql(idx)

me.Controls.Add(new LiteralControl(""))

Dim DDL As New DropDownList

DDL.ID = c.ToString

DDL.DataTextField = vDisplayField

DDL.DataValueField = vCodeField

me.Controls.Add(DDL)

'---Populate the drop down---

Dim mSql As String

msql = vDDLSQL

myConnection = New OleDbConnection(ConnStr)

myCommand = New OleDbDataAdapter(mSql, myConnection)

myCommand.Fill(ds, c.ToString)

DDL.DataSource = ds.Tables(c.ToString).DefaultView

DDL.DataBind

'set the display field

Dim dv2 As dataview

dv2 = new DataView(ds.Tables(c.ToString))

If c.DataType.ToString = "System.String" Then

msql = vCodeField + " = '" + r(c).ToString +"'"

Else

msql = vCodeField + " = " + r(c).ToString

End If

dv2.RowFilter = msql

DDL.Selecteditem.text = dv2(0)(vDisplayField).ToString

DDL.Selecteditem.value = dv2(0)(vCodeField).ToString

me.Controls.Add(new LiteralControl(""))

Else

'-----Text Boxes------

me.Controls.Add(new LiteralControl(""))

Dim Box As New TextBox

Box.Text = r(c).ToString

Box.ID = c.ToString

me.Controls.Add(box)

me.Controls.Add(new LiteralControl(""))

End If

'-----Required field ------

If vRequired.chars(FieldsCount) = "1"

Dim vReq As New RequiredFieldValidator

vReq.controltovalidate = c.ToString

vReq.errormessage= "Please enter " + c.ToString

me.Controls.Add(vReq)

End If

End If

End If

FieldsCount = FieldsCount + 1

Next c

Next r

!*****

INSERT MODE

!*****

Else

For Each c in t.Columns

'Get the field name

vFieldName = FieldNamesArray(FieldsCount)

If len(vFieldName) < 1 then

vFieldName = c.ToString

End If

'Don't show this field

If vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField _

or veditable.chars(FieldsCount) = "0" then

Else

me.Controls.Add(new LiteralControl(""))

'label

me.Controls.Add(new LiteralControl(""))

me.Controls.Add(new LiteralControl(vFieldName))

me.Controls.Add(new LiteralControl(""))

If isDropDown(c.ToString) then

'get the index & other parameters

idx = GetDDLindex((c.ToString))

vDisplayField =DisplayField(idx)

vCodeField =CodeField(idx)

vDDLSql =DDLSql(idx)

me.Controls.Add(new LiteralControl(""))

Dim DDL As New DropDownList

DDL.ID = c.ToString

DDL.DataTextField = vDisplayField

DDL.DataValueField = vCodeField

me.Controls.Add(DDL)

'---Populate the drop down---

Dim mSql As String

msql = vDDLSQL

myCommand = New OleDbDataAdapter(mSql, myConnection)

myCommand.Fill(ds, c.ToString)

DDL.DataSource = ds.Tables(c.ToString).DefaultView

DDL.DataBind

me.Controls.Add(new LiteralControl(""))

Else

'value

me.Controls.Add(new LiteralControl(""))

Dim Box As New TextBox

Box.ID = c.ToString

me.Controls.Add(box)

me.Controls.Add(new LiteralControl(""))

End If

'-----Required field ------

If vRequired.chars(FieldsCount) = "1"

Dim vReq As New RequiredFieldValidator vReq.controltovalidate = c.ToString vReq.errormessage= "Please enter " + c.ToString me.Controls.Add(vReq) End If End If FieldsCount = FieldsCount + 1 Next c End If me.Controls.Add(new LiteralControl("")) me.Controls.Add(new LiteralControl("</Table>")) '----add button Dim AddButton As New Button If mode = "insert" then AddButton.Text = "Add" Else AddButton.Text = "Update" End If AddHandler AddButton.Click, AddressOf AddBtn_Click Me.Controls.Add(AddButton) '-----Validation Summary me.Controls.Add(new LiteralControl("
>")) Dim vSummary As New ValidationSummary vSummary.headertext="There were errors on the page:" me.Controls.Add(vSummary) End Sub Private Sub AddBtn_Click(Sender As Object, E As EventArgs) 'Build the procedure call Dim s As String Dim r As DataRow

Dim c As DataColumn

Dim cell As TableCell

Dim row As DataRow

Dim column As string

Dim Value As string

Dim Fieldscount As integer

Dim vdisplay As string

Dim veditable As string

FieldsCount = 0

If mode = "update" then

s = "Execute " + Updateprocedure + ""

For Each r in t.Rows

For Each c in t.Columns

If vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField _

or veditable.chars(FieldsCount) = "0" then

Else

If isDropDown(c.ToString) then

Dim vdropdown As DropDownList

vdropdown = me.FindControl(c.ToString)

column = c.ToString

value = vdropdown.SelectedItem.value

Else

Dim tb As TextBox

tb = me.FindControl(c.ToString)

column = c.ToString

Value = tb.text

End If

If Value = "" then

```
Value = "NULL"
     End If
     If c.DataType.ToString = "System.String" Then
      If Value = "NULL" then
       s = s + " @" + column + "=" + value + ", "
      Else
       s = s + " @" + column + "='" + value + "', "
      End If
    Else
     s = s + " @" + column + "=" + value + ", "
     End If
   End If
   FieldsCount = FieldsCount + 1
  Next c
 Next r
 s = s + "@" + KeyField + "=" + KeyValue
 me.Controls.Add(new LiteralControl(s))
 RunSql(s)
Else
 s = "Execute " + Insertprocedure + ""
 'Insert mode
 For Each c in t.Columns
  If vdisplay.chars(FieldsCount) = "0" or c.ToString = KeyField _
  or veditable.chars(FieldsCount) = "0"then
  Else
  If isDropDown(c.ToString) then
     Dim vdropdown As DropDownList
    vdropdown = me.FindControl(c.ToString)
     column = c.ToString
```

value = vdropdown.SelectedItem.value

Else

Dim tb As TextBox

```
tb = me.FindControl(c.ToString)
```

column = c.ToString

Value = tb.text

End If

```
If Value = "" then
```

Value = "NULL"

End If

```
If c.DataType.ToString = "System.String" Then
```

If Value = "NULL" then

```
s = s + " @" + column + "=" + value + ", "
```

Else

```
s = s + " @" + column + "='" + value + "', "
```

End If

Else

s = s + " @" + column + "=" + value + ", "

End If

End If

FieldsCount = FieldsCount + 1

Next c

```
s = s + "@" + KeyField + "=NULL"
```

me.Controls.Add(new LiteralControl(s))

RunSql(s)

End If

End Sub

Sub RunSql(vSql as string)

try

Dim s As string

Dim myConnection As OleDbConnection

myConnection = New OleDbConnection(ConnStr)

Dim mycommand As New OleDbCommand(vsql,myConnection)

myconnection.Open()

myCommand.ExecuteNonQuery()

myconnection.Close()

Catch ex As OleDbException

' SQL error

Dim errItem As OleDbError

Dim errString As String

Dim s As string

For Each errItem In ex.Errors

errString += ex.Message + "
>"

Next

s = "
SQL Error.Details follow:
" & errString

me.Controls.Add(new LiteralControl(s))

Catch myException as Exception

me.Controls.Add(new LiteralControl("Exception: " +

myException.ToString()))

End try

End Sub

'-----These Functions associated with DropDown Property------

Function ParseDropDown()

Dim strChar As Object

Dim s As String

Dim j As Integer

Dim count As Integer

Dim idx As integer

count = 1

idx = 0

strChar = Split(DropDown, ";")

```
For j = 0 To UBound(strChar)
```

```
If Len(strChar(j)) = 0 Then
```

Else

```
s = CStr(strChar(j))
```

If count = 1 then

DDLColumn(idx) = s

Elseif count = 2 then

CodeField(idx) = s

Elseif count = 3 then

DisplayField(idx) = s

Elseif count = 4 then

DDLSql(idx) = s

count = 0

idx = idx + 1

End If

```
count = count + 1
```

End If

Next

End Function

Function GetDDLIndex(vDDLColumn as string) as integer

'Pass in the column where a drop-down list must appear

```
' this function returns the row where its details are
```

' stored in the array

Dim i As integer, vKey As integer

```
For i= 0 to UBound(DDIColumn)
```

```
If DDIColumn(i) = vDDLColumn
```

vKey = i

End if

Next

Return vKey

End Function

Function IsDropDown(vDDLColumn as string) as boolean

Dim i As integer, vflag As integer

vFlag = -1

For i= 0 to UBound(DDIColumn)

```
If DDIColumn(i) = vDDLColumn
```

vflag = 1

End if

Next

If vFlag = 1 then

Return True

Else

Return False

End If

End Function

'-----Drop Down Functions end------

Function ParseFieldNames()

'This function parses the FieldNames property

Dim strChar As Object

Dim s As String

Dim j As Integer

Dim count As Integer

If len(FieldNames) <1 then

Exit function

End If

strChar = Split(FieldNames, ";")

```
For j = 0 To UBound(strChar)
```

If Len(strChar(j)) = 0 Then

Else

s = CStr(strChar(j))
FieldNamesArray(j) = s End If Next End Function End Class End Namespace

Project 4 Summary

The GenEditAdd is a very useful control that you can hook up to a DataGrid control and use to edit and insert records. You can replace the built-in editing functionality of a DataGrid with this control. To use the editing functionality of the DataGrid, you need to code a number of events. This is not required in the GenEditAdd control. You set a number of properties of the GenEditAdd control and it carries out inserts and updates in an automatic manner. The insert mode is not available in the DataGrid and the insert functionality has to be handled outside of the DataGrid. Not so in the GenEditAdd control; the edit and insert modes work off the same control in a consistent and similar fashion. This is not all; you can specify drop-down list columns, required-entry columns, and add user-defined column names. The drop-down lists created by the control can access any database table to populate themselves. Further, unlike the ASP.NET DropDownList control, this control maintains state and scrolls down the correct record that matches the value of the field it is attached to. When the user is finished with the insert or update, GenEditAdd calls the appropriate stored procedure to effect the insert or update.

Project 5: Visual Studio.NET

Chapter List

- Chapter 29: Displaying Database Data Using a Strongly-Typed DataSet
- <u>Chapter 30:</u> Writing CRUD Applications with Visual Studio.NET
- <u>Chapter 31:</u> Creating a Web Service Using Visual Studio.NET

Project 5 Overview

Visual Studio brings a set of development tools that enables developers to rapidly build Web- and Windows-based applications in a consistent manner. It is an integrated development environment which enables you to develop applications using a variety of languages, such as Visual Basic, C#, ASP.NET, and so on. Regardless of the chosen language, there is only one environment to understand and use. Visual Studio enhances and extends the functionality originally found in Visual InterDev and applications can be built rapidly using drag-and-drop features. The IDE provides powerful debugging features that work across languages, processes, and stored procedures. In this part, which comprises three chapters, I discuss the important features of Visual Studio.NET, focusing on the various wizards, tools, and components available. Then you will build a database web form using a Typed DataSet and utilizing the drag-and-drop features of Visual Studio.NET. In <u>Chapter 30</u>, you will implement CRUD (create-read-update-delete) functionality in the web form. At the end of Project 5, you will develop and consume web services with Visual Studio.NET.

Chapter 29: Displaying Database Data Using a Strongly-Typed DataSet

Overview

In this chapter, I show you how to bind a DataGrid to a Strongly-Typed DataSet and in the process introduce you to various tools, wizards, and components provided by Visual Studio.NET for database interaction.

A strongly-typed DataSet is a DataSet that is created by Visual Studio.NET. The developer does not write any code for the DataSet and just drags and drops a DataSet object from the Visual Studio.NET ToolBox. A DataSet created in this way binds more closely to the .NET framework than a DataSet scripted manually. The typed DataSet benefits from the syntax completion features of Visual Studio.NET, which can lead to more productive development. A typed DataSet is sub-classed from the base DataSet class and has a schema file (an .XSD file) that describes the structures of the tables the DataSet contains. This schema contains the table and column names, the data types, and information about the constraints on the data. An untyped DataSet, on the other hand, does not have a corresponding schema. Visual Studio.NET automatically generates this schema.

The sample walk-through that follows is a Web Application, written in C#, which displays the data from the authors table of the PUBS database in a DataGrid.

Creating the C# Web Application

Visual Studio.Net organizes application development using the concept of Solutions and Projects. A Solution can have one or more Projects and a Project is a folder that holds together all the objects that make up an application. In this step, I will create a C# ASP.NET Web project. Here are the steps to do it:

- Create a new Web application by either clicking on the New Project button on the Start Page or by selecting File/New/Project.
- Select Visual C# Projects on the left pane and ASP.NET Web Application on the right as shown in Figure 29.1. You can call this project "VSOverView."



Figure 29.1: Creating a new ASP.NET Web Application.

Exploring the Application Folder

VS will create an application folder called VSOverView in the wwwroot folder. It will also create an IIS virtual directory. You can see this by going to the <code>Control Panel</code>,

selecting Administrative Tools, and then Personal Web Manager. Click on the Advanced button of the Personal Web Manager and you will see a new virtual directory called VSOverView as shown in Figure 29.2.

Setting (a)	New Part Weblins	of ange	OX OR DAL	COLUMN A
Dan Santa Sa	Overall P Barth Consulto Spints Consulto Spints de Consulto	2 del Lob Angenies Danie Connike Advances Ad Danie Connike Advances Advances Danie Connike Advances Advances Danie Connike Advan	Control of the second s	an a
Corgonalia -	3.4000 eacherCormond			
H'M.				
Optical Reg			Sol 1 August 1	

Figure 29.2: VS will create a Virtual Directory in IIS.

Exploring the Generated Files

Now take a look at the ASP.NET Web Application created by VS (Visual Studio). Your screen will look like the screen shot displayed in <u>Figure 29.3</u>.



Figure 29.3: The blank project created by VS.

On the right-top side of the screen, you will see the Solution Explorer. You will note that VS has created a Global.asax file, a Web.Config file, and a default web form called WebForm1.aspx. The Global.asax file handles "application level" logic through application events like Application_Start, Application_End, Session_Start, and Session_End, to name a few. This file was discussed in Chapter 10, "ASP.NET Application. Located in the root of an application folder and applicable to all the subfolders in the application folder, this file contains the error, security, compilation, debugging, session, and globalization information. This file was also discussed in Chapter 10. The web form WebForm1.aspx is provided as a template that we can use to create a new web form. Note that at the bottom of the work area of the screen (the white

area), there are two tabs called Design and HTML. You are initially in the Design view. The Design view allows you to drag and drop controls from the Toolbar onto the form. It also allows you to visually position the controls with your mouse.

Web pages generated by VS make use of Code Behind files. If you browse to the VSOverView application folder in wwwroot, you will be able to see the Code Behind file WebForm1.aspx.cs.

To view the Code Behind file in VS, you can adopt either of the following approaches:

- From the Project menu, select Show All Files. Then in the Solution Explorer, you will see a plus sign against the web form WebForm1.aspx. If you click on the plus sign, you will see the Code Behind file.
- In the Solution Explorer, right at the top there is a set of icons. Locate the icon that displays the tool-tip Show All Files and click on it.
- Right-click on the web form WebForm1.aspx and select View Code.

Within the Code Behind please note that various namespaces are already imported. During development, you might create a number of web forms. To view a web form in the browser, right-click on the form name and select View in the Browser. To set a web form as the default form that opens in the browser each time you build and run the application, right-click on the web form in the Solution Explorer and select Set as Start Page.

Exploring the Project Properties

Right-click on the project name in the Solution Explorer. The Property Pages window for the application displays as shown in Figure 29.4.



Figure 29.4: The Project Properties.

You can specify the namespace of your project in the Default Namespace box. This, by default, is your application name. If you change the namespace here, the namespace change will not be automatically applied to already created forms. You must manually make the namespace change to all the pre-created web forms, else you will get compilation errors. Any new web form will be created in the specified namespace. At this stage give a meaningful name to the WebForm1.aspx. Call it TypedDataSet.aspx by right-clicking it in the Solution Explorer and selecting Rename. At this stage if you open the Code Behind file and note the class name and the constructor name, you will see that even though you changed the name of the form to TypedDataSet, the Code Behind file still refers to WebForm1. If you do not fix this, you will get build errors later on when you add components to the form. To fix this, in the Code Behind file, change the class name and the constructor name to be TypedDataSet and you should be ready to

go. You can do a find and replace or now that I have made my point, just delete and recreate the web form with the new name.

A constructor is a method that has the same name as the class name. This is the first method that gets fired when a class is initiated (a web form is a class). A default constructor, which has no parameters is associated by default with a class. You can, however, provide additional constructors that accept parameters.

Database Interaction

In this section, I will show you how to connect to a database, populate a DataAdapter, and use it to fill a DataSet. I will use the authors table of the PUBS database for the demonstration. I will show you two alternative methods of connecting to the database and populating the DataAdapter.

All the components that interact with the database are to be found in the Data section of the toolbox. To use any component, all you have to do is to drag and drop them onto a web form. Figure 29.5 shows the components that are available for database interaction.



Figure 29.5: The Data section of the toolbox.

Creating a Connection and a DataAdapter

I will use the SQL Managed Provider to interact with the database. As you might remember, the SQLConnection is used to connect with the database. A SQL Select Query is applied against this connection to fill the SqlDataAdapter with database data. As I promised, I will show you two methods to achieve this.

Creating a SQLConnection and a SqlDataAdapter - Method #1

This method of creating a SQLConnection and a SqlDataAdapter involves dragging and dropping a SqlDataAdapter from the Data tab of the toolbox onto the web form and completing a number of wizards. This is how to do it:

 From the Data tab of the toolbox, drag and drop a SqlDataAdapter onto the web form. The DataAdapter Configuration wizard starts which you will use to create the connection and command. Click on next and you will see the Choose your Data Connection window. Click on the New Connection button and you will see the Data Link Properties window.

2. The window should be filled out as shown in Figure 29.6.

Set the server name, username, and password in this pane. If you left the pubs database at its default setting, the username/password would be

sa/blank. Check the option box Allow saving of password. Finally select the pubs database in the Select database on server box and hit on the Test Connection button to test the connection.

: Type:DutzTet - Minsosit Ves	al Fasic	NET John	ipe) - Wald	ional anger	•						A9
to Life time front Ball	(pelug	EgsCon	Fynet	Tata in	et Deb	a Madae			-		
5. 1. 0. M 0 1 4	a .		411.10	a septo					· 20.86.3	. 6	7.
- 0 · - 0 ·							H 7 U	A 2 3		1= 11	au.
10100	1 Perce	- include	on relation for	wheel as	- 1444	and sup!	Mel/ant-d/			10	NAME OF TAXABLE PARTY AND ADDRESS.
was Fame	10				0					21	
Data		(an in	me fee	and plan	addre	advity)mat	eletip (reastern				Solden TypedColder (1 print)
A Posta	abs	abs	abc	abo	abc	abe abe	abc False				a of the second
L faster	abc	abe	she	abe.	abe	abe abe	abo True				Covig-mb
14 SG Connection	abc	abc	alsc	abc	abc	abc abc	abc False				a driven with
(1 ACCenterion	abe.	abe	de	de la	de	abe abe	abo True				A) Shimon
'S Endeninat	abe	abe	abc	abo	abe	abc abc	abc False				- E Spediatel et deux
10 Lode	Pate 1	abe	40	100	de	abe abe	abo True				C) Webf card acps
G ADD date Canadi	and the	abe	ake.	alte.	ake.	abe abe	ahc False				
(j) 10.1mdeCorpet	abs.	abe	ale	44	abe	abe abe	abe True				
	1.00	der .	de la	de la	de .	also also	also False				
	100	-	40	40	de	also also	de Tex				
	1.00	and a	44		de la	also also	also False				
	1000	-	-	-			A	- e -			
							Long-training				6
							- Intuin	di futto	INPUT VINCINGS	(analas	· ·
	194	00404	Central 1	13 4000	method	(Grann	LA -F IND	-2 5			
LONGO -	Dates						Sec. 1				-
Calcord Time	11120						tight -		Test		
Secol	10.04	an D	FHL				0.000				-
CTR AND IN CO.	-	-	_				144	-drugs			
Louise Labor							E4.0	AUROP .	20-staffeet	-	-1
							a a start a st		- Baller		
							10000		Carl and a second second		
Build 1 currented Diplot B	Signed .						and the second				
Dalatin Conservation Conservation	a a casa						The Date	distantia salad	In course to he have	-	1. Ter
							and/or				
-							-	_		-	
After spide accessed									168	K.	Ca19 0-18
melone lone f	on Tax	One	diame.	(Witer)	of these	of Street	1	in the	OR OTACLE	1.0.77	CARLON BURNALL

Figure 29.6: The DataLink properties pane.

3. In the step, the wizard presents you with three options for selecting the query type; Use SQL statements, Create new stored procedures, and Use existing stored procedures. This is shown in Figure 29.7. As I will discuss shortly, an insert, update, delete, and select command can be associated with a DataAdapter. The Use SQL statement will create four SQL statements, one each to select, insert, update, and delete rows from the database. Similarly, the Create new stored procedure option will create four new stored procedures and the Use existing procedure option will enable you to supply the names of four pre-existing stored procedures. For the purpose of this demonstration, choose the Use SQL statements option.

Second Image: Second	Calcul Nex	Villiam Page (Welford and Welford	and, she	- 0 X	CR D M m
Place The JOO Concept Local Clarge () End the The Job Concept Local Clarge () End the Phase The Job Concept Local Clarge () Phase Phase	irred -	Stabfeed (rpedbataSet)	 I', (Pecharations) 		Solden Typed utiler (1 point
- For two	A Face	No. 1007anapertient - End Yam - End Yam - Bolits Paratien Tydes - No. 1007ant Johnson - Rossin spiderellist - End Paratien - End Parati	.Cope() eductioner()()(), updeceduction, in Syn soli Spote from ()) (united Commonly and Spote cher about o (updated ber about about + () eductaber.)))))))))))))	zi rom.br	 Top Topped Default Topped Defaul
		- End 5x8			
		•			

Figure 29.7: The Choose a query window.

4. A new window appears which is called Generate the SQL Statement. This window allows you to type in a Select query. You can type in the query Select * from authors as shown in Figure 29.8.

العراد والاستر				Herverger	1.00
	vitifient apr		_		1 5.0
n france	phann address	rity	110	dip interview	1
ohneon	408-496-7223 10932 Bigge B.4	Mealo Fark	CA.	94025 Teur	
lajore	415 986-7920 309 6364 24 #411	Osidad	CA.	94638 Teur	
Bergi	415 548-7723 589 Darwin Ln.	Beckeley	CA.	34705 Teur	
fabel	408 286-2428 22 Circeland Av. #14	SanJuse	CA	95128 True	
least.	415 134-2919 5420 College Art	Calderd	CA.	946609 Titue	
(easier	913 843-0462 10 Massroppi De	Lostence	305	64044 Fahr	
Uraham	415-658-9932-6223 Bateman 28	Berkeley	CA.	94705 Teat	
lin .	415 136-7128 3410 Binute 31	Taio Abo	CA	34300 Teur	
hart .	707 138-6445 PO Box 792	Contis	CA.	95428 Teur	
halese	415 585-4620 18 Broadway Av.	San Francisco	CA	94130 Teue	
formingstar	615 297-2723 22 Graphar Brast Rd	Hadride	TOT	37215 False	
- blongs	503 745-6402 55 Hills-falle BI	Cornda	08	97330 Tese	
like -	415 935-4228 3 Salver Ct	Walnut Creek.	CA.	94585 Taue	
met.	615 396-8275 2286 Crim PL #65	Ann Arbor	M	48105 Teur	
Gibil	219 547-5902 3 Babling 71	Gary	10	46400 Taue	
Wk.	415 843-2991 5429 Tolograph Arc	Osidand	CA	34509 False	
leaves	435 354-7128-44 Upland Sh	Osidani	CA.	94612 Tese	
ana -	415 534-9219 5720 McAuley D	Guidand	AD.	94609 Teur	
lybria	301 946-8853 1956 Arlegtus 7t	Lockade	MD	20850 Teur	
lead	415 136-7120 3410 Bloods Dt	Talo Alto	CA.	94001 True	
Seather	707-648-4982 301 Putners	Vacuelle	CA	95688 False	
lane	801 826-0752 67 Seventh Av.	Salt Lake City	UT.	04152 Titue	
thet	801 826-0752 67 Seventh Av	Sab Lake Cay	UT	84152 Texe	
	Abane darjon farjon farjon farjon farjon farjon farjon farjon an an an an an an an an an an an an an	Battern Mathewa Mathewa <t< td=""><td>Battan Jataba Mathematical Control (1997) 0.1 Mathematical Control (1997) Mathematical Control (1997) 415 Sile - 7220 Sile Sile (1997) Mathematical Control (1997) 415 Sile - 7220 Sile Sile (1997) Mathematical Control (1997) Mathematical Control (1997) 415 Sile - 7220 Sile Darwin La Berlarly Mathematical Control (1997) Mathematical Control (1997) Mathematical Control (1997) Sile 2010 Sile 2010</td><td>Batura Jatura Jatura Johney Ope State anne 4019 (47-7221 1992) Exp Hold of Link CA CA dergine 4019 (47-7221 1992) Exp Hold of Link CA Ca dergine 415 (340-7202) S00 Chiel & H11 CA Darkadel CA hand 415 (340-7202) S00 Chiel & All Link Darkadel CA hand 415 (340-7202) S00 Chiel & All Link Darkadel CA hand 415 (340-2019) S20 Chiel & All Caldael CA hands 415 (340-7202) 200 Chiel & All Caldael CA hands 415 (340-7202) 202 Datersan St Darkadel Ca hands 415 (340-7202) 200 Han 792 Centle Ca hands 415 (340-6203) Sildeble B Centle OB hands 415 (340-6203 Sildeble B Centle OB hands 415 (340-6203 Sildeble B Centle OB hands 415 (340-6203 Sildeble B Centle OB hand 415 (340-72013</td><td>Barriso Barriso <t< td=""></t<></td></t<>	Battan Jataba Mathematical Control (1997) 0.1 Mathematical Control (1997) Mathematical Control (1997) 415 Sile - 7220 Sile Sile (1997) Mathematical Control (1997) 415 Sile - 7220 Sile Sile (1997) Mathematical Control (1997) Mathematical Control (1997) 415 Sile - 7220 Sile Darwin La Berlarly Mathematical Control (1997) Mathematical Control (1997) Mathematical Control (1997) Sile 2010 Sile 2010	Batura Jatura Jatura Johney Ope State anne 4019 (47-7221 1992) Exp Hold of Link CA CA dergine 4019 (47-7221 1992) Exp Hold of Link CA Ca dergine 415 (340-7202) S00 Chiel & H11 CA Darkadel CA hand 415 (340-7202) S00 Chiel & All Link Darkadel CA hand 415 (340-7202) S00 Chiel & All Link Darkadel CA hand 415 (340-2019) S20 Chiel & All Caldael CA hands 415 (340-7202) 200 Chiel & All Caldael CA hands 415 (340-7202) 202 Datersan St Darkadel Ca hands 415 (340-7202) 200 Han 792 Centle Ca hands 415 (340-6203) Sildeble B Centle OB hands 415 (340-6203 Sildeble B Centle OB hands 415 (340-6203 Sildeble B Centle OB hands 415 (340-6203 Sildeble B Centle OB hand 415 (340-72013	Barriso Barriso <t< td=""></t<>

Figure 29.8: Typing in a SQL query in the Generate the SQL Statement window.

You can also use this window to graphically build a SQL query. To do this, click on the button captioned Query Builder. This will display a query builder tool as shown in Figure 29.9. This is a Microsoft Access type query builder tool that you can use to select and join multiple tables and then drag and drop column names to build the query. I leave it up to you to explore it in detail.

Signal defau Monard Mariel (Mariel (Mariel)) additions and De (All Sone band (Ball (Ball) (Sone) Sone (Ball (Ball) (Sone) (C + 1) (S ⊂ (B + 1)) (S ⊂ (C + 1)) (Sone) (S ⊂ (C + 1)) (C + 1) (S ⊂ (C + 1)) (S ⊂	· 32**· 0.
Cold Mit Steres Cold Mit S	to a second
Seed - States 3.1%.	<u>ت</u> ر
Foreine funder 	

Teal) 創作は「Qual-antenes」Qual-antenes」(2) House View (2019)(2010) (2019)(2010) (2010) (2010) (2010) (2010) (2010) (2010)

Figure 29.9: The Query builder allows you to graphically build queries.

- Finally hit Next and then Finish after you are satisfied with your Select query.
- A SqlDataAdapter and a SqlConnection will appear at the bottom of the web form as shown in <u>Figure 29.10</u>. These will be called SqlDataAdapter1 and SqlConnection1.

10 × [] 0		in in "
Con Pase Pase Casod Casod Chickasette Chickasette Chickasette Scianette Scianette Scianette	Sectory Pareliation Pareliation 0.1.1 0.1.1 Pareliation Pareliation Pareliation 0.1.1 Pareliation Pareliation Pareliation 0.1.1 Pareliation Pareliation Pareliation 0.1.1 0.1.1 Pareliation Pareliation Pareliation Pareliation 0.1.1 0.1.1 Pareliation Pareliation Pareliation Pareliation Pareliation 0.1.1 <	3 Abort store 100 (100) 3 20 (100) 100 (100) 4 Micro Volenven (1) 100 (100) 5 Micro Volenven (1) 100 (100) 6 Micro Volenven (1) 100 (100) 6 Micro Volenven (1) 100 (100) 6 Micro Volenven (1) 100 (100) 7 Micro Volenven (1) 100 (100) 8 Micro Volenven (1) 100 (100) 9 Micro Volenven (1) 100 (100) 9 Micro Volenven (1) 100 (100) 9 Micro Volenven (1) 100 (100)
LAurus - gonwen 6. Load Tang mai	Sufacestates	
9-7		2
		(HynametProperties) May value in the application configuration the facad

Figure 29.10: The SqlDataAdapter and SqlConnection created in VS.

This is the outcome that we sought in this first method of creating a SqlDataAdapter and SqlConnection with Visual Studio. Now we examine method #2.

Creating a SqlConnection and a SqlDataAdapter - Method #2

I am now going to show you another method of creating a SqlConnection and SqlDataAdapter that is much quicker than method #1. You need to start with a clean slate, so add a new web form to the project and call it TypedDataSet2.aspx. Since you will need to view this form in the browser, set this form as the start-up page by rightclicking the form in the Solution Explorer and selecting Set as Start Page.

The Server Explorer

The Server Explorer is a database explorer that shows the various database objects like tables, triggers, and stored procedures. It is accessed from the View/Server Explorer menu option or the shortcut Ctrl-Alt-S. The Server Explorer is shown in Figure 29.11. You can use it to manipulate database tables. I leave it to you to explore it in detail.

Bit Bit <th>THE ROOM COMPANY AND ADDRESS</th> <th>and a state of the state of the</th> <th></th> <th></th> <th></th> <th></th> <th></th>	THE ROOM COMPANY AND ADDRESS	and a state of the					
1 20.3 Line 10.3 Line Annuel 10.3 Line Annuel <td< th=""><th>Lange Contraction of the Contrac</th><th>NETWORPHENE </th><th>db0.authors (</th><th>In here</th><th>Lou loope</th><th>In d</th><th>2.0.2.1</th></td<>	Lange Contraction of the Contrac	NETWORPHENE	db0.authors (In here	Lou loope	In d	2.0.2.1
	Parts The Astroney Million State	10032 Bage Nd. Plerio	408 496 7227	Menor	10.040	▶ 172-33-1071	Co. / a bit forcersi
Benericht ist Benericht ist jernetig Benerichterichtist jernetig Benerichtericht ist jerneti	1 Calue - D. Million Home	20143-030 R. P411 Callar	425 996 7900	Markine	Green	212-41-815	in the second
8 8 8 8 8 8 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 8 9 9 8 9 9 8 9 9 8 9 9 8 9 <td>Bertall II. Likebarrante</td> <td>Stricesenics, Berlan</td> <td>495.040-7723</td> <td>Change</td> <td>Caren</td> <td>280-03-0366</td> <td>in the number of the</td>	Bertall II. Likebarrante	Stricesenics, Berlan	495.040-7723	Change	Caren	280-03-0366	in the number of the
a transmitter in state in the state in th	A Sec A	22 Geneland Av. # San A.	408.256-2408	Hoted	014915	287-49-2384	 A departure
A Training of the second se	Color if Americante	SUNCERSAL CASE	101011-015	210	20.001	274.81.4094	in the sector
1 1 <td>Laca in the data was</td> <td>TOPostogo Ch. Lawren</td> <td>1010404062</td> <td>- New der</td> <td></td> <td>- 24-1-126</td> <td>a la mare</td>	Laca in the data was	TOPostogo Ch. Lawren	1010404062	- New der		- 24-1-126	a la mare
3 The stand of	Brief B. St. Conductor	100 Datamar S. Berlar	425 000-0002	Apraham	Devid	400-01-7000	
S	And Channels	ALCOHOL S. FILLS	100000-000	Aut.	diam'r.	477-17-1217	a di manual
Source failer (Entrementer), 5 falled, 5 drigged	Lab a distance	Differences in Section	100 000 0000	Charles	Condition of the local division of the local	and the lot of the	a la second
A Standard Barran A Standard Ba	The second second	Titledge in a line	100 100 1000	Manager 1	Contract of	100 CT 1/10	
	Cross Constant	White Cros	811 Tell and 1	Longitude .	Burberry .	445-61-1470	and the second s
Text of the second of centre in the second of the se	Table	1 Ter (). Water	108 108 400	1.00	Training .	470.71.5740	a approximation
Image: Processed of Status Status Main District Main District Status Main District District Main District Nain District District District District Dis	A Ann A	2018 Cancel 415 Aur &	105.996-8275	in the second	old Castilla	712-45-1002	10 million (100 million)
Altre Calue Constant, 5 failed, 5 skippel	dev.	Melton, Gev	215547-0002	Hille	dufeator	722-01-0404	10.02 enters
Allen Balen	- Ofer	SKI SROAMA OKA	405.045-2948	245	Trapy	124-03-1908	- () a) a
Annue Colore De Carles, 5 failed, 5 delaged	Calla Constitution	estated its. Collar	405 354 7120	Seam	Hafvaller	724-03-006	C without
A der and a der	Odde Characterian	S700 Yelkaley St. Oddar	44513145237	10.00	Karsen	796-31-1296	- C) at 71000
Alter Calculate & Statistic & Statist	Rady	105 Adapter H. Radu	301946-8803	2 Auto	Partney	N246-10-080	C) by a
A meet Calce 2 meeters 5 failed, 5 skipped	Aux 10 31 8 88	Hildenik S., Pak-A	105 834 7128	Seri	market	846-53-7186	- [] A010
bine faine menose bine faine menose bine faine bine faine faine bine faine faine fai	Yathe	30 Putral Victor	207440-4902	Hoter	Heliaton	000-73-4158	E (8)
Den and the set of the	Settle	17 Secondluker Sell La	10110344752	Arra .	Angel	090-41-2025	- () state
C other C other S or of Color S of Color S of Color S of Color S of Color S of Color S of Color S of Color S of Color S of Color S of Color S of Color S of Color S of Colo	for Li	47 Secold Ac. Sol Li	801 806-8792	Abot	Anger	998-73-3567	E) #
Non-R Calcon Non-R Calcon Non-R Calcon Annue Calcon Non-R Calcon Non-R Calcon Ad Non-R Calcon Non-R Calcon Paldó 1 recenteded, 0 shigget Non-R Calcon Non-R Calcon Paldó 1 recenteded, 0 shigget Non-R Calcon Non-R Calcon						*	C other
k region * * * * * * * * * * * * * * * * * * *							8 - C Annuals
h transficker (betreen state of the state of							H- C replace
All	and the second se					white the second	Same Labor 40
Pri and Table 1 secondaria, 0 deligned Table 2 and 2 a						- Julia	Testadores P
Ad I I annovadori, 5 failed, 5 delapari Supiry: 5 matematica, 5 failed, 6 delapari Supiry: 5 matematica, 5 failed, 7 delapari							<u>tyd</u>
Polifi 1 seconder, 0 failet, 0 driget Septer 2 merendet, 0 failet, 0 driget 2							
Politi 1 researched, 5 failed, 6 deligent Jupier: 8 menesets, 5 failed, 8 deligent 21							
Totor - Remain - Thirty - Report	- 24						
	- 12						Sector & secondary, a factor, a
							solution a secondary a consta-
E	16						
		<u> </u>					
	1						La contra de

Figure 29.11: The Server Explorer allows you to explore and manipulate database objects. Now all that you have to do to create the SqlConnection and the SqlDataAdapter is to drag and drop the authors table from the Server Explorer onto the form (you can refer to Figure 29.11 to locate the table in the Server Explorer). The resultant form looks the same as in Figure 29.10 and has both a SqlConnection and a SqlDataAdapter called SqlConnection1 and SqlDataAdapter1 respectively.

Exploring the Code Generated for the SqlConnection and SqlDataAdapter

If you right-click on the SqlConnection1 and select Properties, the property page for this object displays as shown in <u>Figure 29.12</u>. Note that the Name and the ConnectionString can be modified in this window and this tool enables you to modify property settings through property pages like Visual Basic 6.



Figure 29.12: The Properties page for the SqlConnection.

Similarly you can view the property pages for the SqlDataAdapter as shown in Figure 29.13.

1. VIC-office -Microsoft Vec	d Car At 1 (design) - Type (Datafiet anys									
the talk give proved that	d gabes light report liabs poort report 3	ste under mit								
13-13-020 21	Real and a constant of a code of the	de Lastanete : 3.2223.0.								
ENR ALIG I		J JA / B A / B B B B B B B B B								
All and a stand of the	A 12 DESCRIPTION OF TAXABLE PARTY.									
Notice .	• A BELLE HARDEN STREET ARTICLE Typedo	stature argue	a the last							
1.00	1-9	1 년3	8 J 4 19 1-							
Piece			skon VSD-even () practs							
S cases	characters		<u>ات</u>							
2 ² ceccentrates	selletableptert InterablesIdDet.JoDate	ulujher	-							
3, Occurrentian	DNEVE		Mars							
C ceccened	III (Scharaffregerfield)									
S STRANGE	(Nerwi)	without and approximation of the second seco	ant o							
& Statements	AcceptChargeduregHi	bu .	ALC: NOT THE OWNER.							
V XXX MARK	All (skeiCoanard	sglieletet severandi	basic sp. 1							
All Calation	M PronCommand	oglaser(Cananuad)	cales.							
	Humph Insulting	AN								
	Holders.	Panke								
	III Selectorment	selficientEstamandi								
	Låsdlapsrap	(Collection)								
	HI UpdateContrand	subjudatie:Command1								
	Canfrans John Adapter Special Linksed Top	Cardness Soci Adazon								
	Defailed command used starty updates for detections in Detailet.									
	1	2								
with rising	Parameter Parameters									
100 million and 100 million an										
Celoseffice			21.0							
General	Gimp Circles	1								
Anaty .		and the second se	Concernance of the							
ABCER NORS	シャルションのないに、	1000 Jan 173 Jan 800	SO34 INT							

Figure 29.13: TheProperty Page for the SqlDataAdapter.

Now let's take a look at the code that was generated. Bring up the Code Behind file in VS and expand all nodes (click on the plus sign before various methods and the sign will become minus. You will now be able to see all the code of that method). Note the #region and the #endregion tags. Code enclosed within these tags allows you to specify a block of code that you can expand or collapse. You can also browse to the application folder and open the Code Behind file TypedDataSet.aspx.cs in a text editor to view the generated code.

Various namespaces are imported in the Code Behind file as follows:

using System;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Web;

using System.Web.SessionState;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Web.UI.HtmlControls;

A class called TypedDataSet, which inherits from System.Web.UI.Page is created in the VSOverView namespace as follows:

```
namespace VSOverView
```

```
{
    public class TypedDataSet : System.Web.UI.Page
    {
        //A lot of code here
    }
}
```

Four Command objects are declared as follows:

[code]

protected System.Data.SqlClient.SqlCommand sqlSelectCommand1; protected System.Data.SqlClient.SqlCommand sqlInsertCommand1; protected System.Data.SqlClient.SqlCommand sqlUpdateCommand1; protected System.Data.SqlClient.SqlCommand sqlDeleteCommand1;

A SqlConnection is declared as follows:

protected System.Data.SqlClient.SqlConnection sqlConnection1;

A SqlDataAdapter is declared as follows:

protected System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1; An instance of all these objects in creating using the new keyword in the InitializeComponent method as follows:

this.sqlSelectCommand1 = new System.Data.SqlClient.SqlCommand();

this.sqlInsertCommand1 = new System.Data.SqlClient.SqlCommand();

this.sqlUpdateCommand1 = new System.Data.SqlClient.SqlCommand();

this.sqlDeleteCommand1 = new System.Data.SqlClient.SqlCommand();

this.sqlConnection1 = new System.Data.SqlClient.SqlConnection();

this.sqlDataAdapter1 = new System.Data.SqlClient.SqlDataAdapter();

In the same method, the SqlConnection is set as follows:

// sqlConnection1

this.sqlConnection1.ConnectionString = "data source=BHASIN\\NETSDK;

initial catalog=pubs;integrated security=SSPI;persist " +

"security info=True;workstation id=BHASIN;packet size=4096";

The DataAdapter is associated with a select, insert, update, and delete Command object as follows:

this.sqlDataAdapter1.DeleteCommand = this.sqlDeleteCommand1;

this.sqlDataAdapter1.InsertCommand = this.sqlInsertCommand1;

this.sqlDataAdapter1.SelectCommand = this.sqlSelectCommand1;

this.sqlDataAdapter1.UpdateCommand = this.sqlUpdateCommand1;

Now for each of these Command objects, the following properties are set:

- The CommandText property
- The Connection property and
- The Parameters collection

Though there is a lot of code, it is basically very simple. Most of the code involves setting the parameters collection. Each field of the database table is added as a parameter to the parameters collection of the Command object and an insert, update, or delete query is built which includes all of these fields. I have explained this technique in relation to calling a stored procedure with a Command object in <u>Chapter 3</u> (Using ADO.NET in the .NET Framework) which you might like to refer to now.

It might be proper to talk about the relevance of these four commands here and also bring the DataSet into picture. In ADO 2.x, if you needed to perform a select, insert, update, or delete all in one method, four ADODB command objects would be required. In ADO.NET there are still four commands but all are linked to a single DataAdapter object. The DataAdapter uses any of these four command objects as appropriate. I will incorporate the DataSet in the <u>next section</u> but this is an appropriate juncture to discuss it. As you may recall a DataAdapter sits between the DataSet and the database. The DataSet in turn may be bound to a list control like a DataGrid. The user interacts with the DataGrid and may change database data. Since the DataSet is bound to the DataGrid, any changes to the DataGrid are also made to the DataSet. You could provide a save button on the DataGrid that the user could click if he wanted to save changes to the database. These changes could be inserts, updates, or deletes to database rows. This button would call the update() method of the DataAdapter. The update() method checks the DataSet for all inserts, updates, and deletes and calls the appropriate update, delete, or insert command as appropriate. Note that the update() method of the DataAdapter performs the entire insert, update, and delete activities, in one go (and not just the update activity, as its name seems to indicate).

Finally the TableMapping property provides friendly names to refer to the database column names. Here the name and the friendly names are the same. For example the TableMapping for the au_id field is as follows:

new System.Data.Common.DataColumnMapping("au_id", "au_id").

This property links the names of the database columns in the DataSet with the database. Thus, if you have a very long database column name, you can refer to it with a short alias in the DataTable.

DataSet

In this step, I will create a Strongly Typed DataSet using the drag-and-drop features of Visual Studio.Net. I will add some code to populate the DataAdapter and the DataSet and then bind a DataGrid to the DataSet. Here are the steps to do it.

- 1. Select the Generate Dataset menu option from the Data menu.
- 2. The Generate DataSet dialog box appears as shown in <u>Figure</u> <u>29.14</u>. Call the DataSet dsAuthors and ensure that the box that says Add this DataSet to the designer is checked.

Balance Beneric B	Sectors			Statu Exten : (Source) Statu Exten : (Source) Statu Exten : (Source) Statu : (Sourc
Opticanditing Garana Output	Giteopi Det	07 Groot Not	• *	

Figure 29.14: The Generate DataSet dialog box.

3. An instance of the DataSet called dsAuthors1 appears at the bottom of the form and the object dsAuthors.xsd displays in the Solution Explorer. This is shown in Figure 29.15.



Figure 29.15: An instance of the DataSet is created. by VS.

If you note the code generated in the Code Behind file at this stage, you will observe the following:

A protected variable dsAuthors1 of type dsAuthors is declared as follows:

protected VSOverView.dsAuthors dsAuthors1;

An instance of dsAuthors called dsAuthors1 is created using the new keyword as follows:

this.dsAuthors1 = new VSOverView.dsAuthors();

The DataSetName, Locale, and Namespace properties for dsAuthors1 is set as follows:

// dsAuthors1

this.dsAuthors1.DataSetName = "dsAuthors";

this.dsAuthors1.Locale = new System.Globalization.CultureInfo("en-US");

this.dsAuthors1.Namespace = "http://www.tempuri.org/dsAuthors.xsd";

- 4. Now drag and drop a DataGrid from the Web Forms Toolbox onto the web form. In the next step, I will bind this DataGrid to the DataSet.
- 5. In this step, create the Bind() method which will populate the DataSet using the Fill method of the DataAdapter and bind the DataGrid to the DataSet as follows:
 - 6. public void Bind()
 - 7. {
 - 8. sqlDataAdapter1.Fill(dsAuthors1);
 - 9. DataGrid1.DataSource=dsAuthors1;

}

Finally call the ${\tt Bind}$ method in the ${\tt Page_Load}$ event as follows:

private void Page_Load(object sender, System.EventArgs e)

{

if (! IsPostBack)

```
{
Bind();
}
```

}

Visual Studio.NET has wizards to build methods and properties, which I leave to you to explore. Typing Ctrl+Shift+C or selecting View/Class View will bring up the Class View window. This view lets you explore the methods, properties, and fields of the class. If you right-click on the class name and select Add, you will be able to access wizards that help you create a method, property, indexer, or field.

10. Build and run the form by pressing F5. <u>Figure 29.16</u> is the screenshot of the resulting output.

piller de la	4,993-414	and codes dot ser									
4.0.214	() ton 1	ston Cleante S	Contra the	orna alla	happing ma	Auton Chikok	of Caleron				
- 33	er id	as have	as Barte	phone		different .	otr	inter la	120	contract	
	172-32-1	170 White	Johanne .	408-496-1	1223 8	1912 Digge Rd.	Mende Park	CA	94025	True	
1	213-46-3	P15 Oren	Marjora	415 986-1	10.20 3	1174 Std St. #11	Oakland	CA	94618	The	
	298-95-7	THE Carson	Curol	415 548-1	1723.5	19 Dawie Le.	Berkeley	CA	94705	Tree	
	267-43-2	1940Leay	Mithael	408286-2	M29/2	Cleveland Av. #14	San Jone	CA	95128	The	
	274-86-5	391 Straight	Dem	415 834-3	9195	Of Codege Art	Oakland	CA	94605	True	
	341-22-1	762 Seath	Measter	513 043-0	1462.5	Manimippi Dr.	Laurence	2.5	66044	False	
	409-56-7	OCE Resort	Abrohum	415 658-5	1132.6	229 Bateman St.	Beckeley	CA	94765	True	
	427-17-2	219 Dvil	Ann	415 036-3	120.3	410 Disade St.	Pale Alto	CA	94303	True	
	472-27-2	340 Gringlesby	But	707 938-4	4452	O Box 792	Cerelo	CA	95428	The	
	486-29-1	706 Locksley	Chaine	485 585-4	6201	Elenadory Ar	Sas Francisco	CA	94130	True	
1.1	527-72-3	Q46 Greece	Morragete	615 297-2	17232	2 Oraybar House 3:4	Nashville	TH	37215	Faire	
	648-90-1	272 Rindet Hale	Regeald	503745-4	6400.5	5 Hille date 30	Consilis	CR	97330	True	
	(72-71-3	249 Yokomete	Akko	415 935-4	1220.3	Silver OL	Walest Creek	CA	94585	True	
	112-45-1	SIT del Cantillo	Innes	415 591-1	2752	284 Oran P1 #86	Ann Arbor	M	48165	True	
	722-51-5	454 DeFrance	Michel	219 547-0	1023	Dailing FL	Gay	24	46403	True	
	724-08-9	933 Striger	Dek	415 843-3	8915	Of Telegraph Av.	Oakland	CA	94009	Fale	
	724-86-5	391 MacFeather	Seams	415 354-3	1128-4	4 Upland Hts.	Oakland	CA	94612	True	
- 13	756-36-7	191 Earm	Livia	415 534-1	0195	720 McAuley DL	Oakland	CA	9460	True	
	807-91-6	454 Pattoley	Sylvia	301 946-1	III 53 P	256 Adaption PL	Rockville	MD	20853	True	
- 13	\$46-52-7	186 Eluster	Stept	415 836-3	1283	410 Bloude St.	Pain Albo	CA	94301	Tree	
	EK3-72-1	151McBaklen	Heather	207448-	002.3	01 Patean	Vacandle	CA	95688	False	
	299-46-2	1005 Ringer	Anne	801 824-4	1752-6	7 Seventh Are	Sid Like City	UT.	84153	Time	
	\$48.71.1	1567 Binner	Aber	801 824-0	1752-6	7 Seconds Acc	Salt Lake City	TU	84152	True	

Figure 29.16: The final output.

You will appreciate that Visual Studio.NET allowed us to present database data in a DataGrid with only a few lines of code. As a RAD tool, Visual Studio.NET handles well. The code it generates is elegant and concise and using it could cut down your development time quite a bit.

Chapter 30: Writing CRUD Applications with Visual Studio.NET

CRUD is the acronym for Create-Read-Update-Delete, which are the four basic functions of database interaction. Visual Studio.NET goes beyond building pretty screens and nifty graphical user interfaces. As you will see in this chapter, you can use Visual Studio.NET to build a powerful database application that has functionality to select, insert, update, and delete database data.

In this chapter, I will build a web form which will display records from the stock_master table. This form, which will be built using Visual Studio.NET will provide functionality for the addition, modification, and deletion of database data.

Create a New C# ASP.NET Web Application

As always, you need to create a new ASP.NET Web application. I have chosen C# as my scripting language. Here are the steps to create the new application.

1. Create a new Web application by either clicking on the New Project button on the StartPage or by selecting File/New/Project.

- 2. Select Visual C# Projects on the left pane and ASP.NET Web Application on the right. I have called this project Chapter30. Delete the WebForm1.aspx that is created. Add a new web form and call it StockMaster.aspx.
- 3. Select a BackGround color for StockMaster.aspx by right-clicking on a blank portion of the web form and selecting a Background color from the Color and Margins tab of the Document Property Page as shown in Figure 30.1. I have selected the color #99cc99.



Figure 30.1: Setting the BackGround color of the web form.

- 4. Add the SqlClient namespace to the Code Behind file. Visual Studio.Net automatically adds all the required namespaces except for the Sql Managed provider. Add this namespace as follows:
 - 5. using System.Data.SqlClient;

The namespace section will now look like this:

using System;

- using System.Collections;
- using System.ComponentModel;
- using System.Data;
- using System.Drawing;
- using System.Web;
- using System.Web.SessionState;
- using System.Web.UI;
- using System.Web.UI.WebControls;
- using System.Web.UI.HtmlControls;
- using System.Data.SqlClient;

The Data Components

Now, add a SqlConnection, a SqlDataAdapter, and a DataSet as follows.

 Create the SqlConnection and the SqlDataAdapter by dragging and dropping the stock_master table of the ASPNET sample database (installation instructions can be found in <u>Appendix A</u>) from the Server Explorer onto the form. The Server Explorer will show the stock_master table as shown in <u>Figure 30.2</u>.

a Lat then Parent Rull Other	Former Lidie Board Frances Links Minday 1948	1913
	THE HAS I DOWN I DIRE . O	a
Implait		
un Tuden 2 1	and the second se	Same Property Chantanits # 1
0.3.11	Bandar dissertantic official and point and age	
************************************	¹ *** papers, we write git a ray is hand risk, and institution and an exception of the second rate, Second Control (constrainty). The diffusion of the second papers of papers of the second rate of the second r	 Baller Construction (19) Baller Construction
115.		() () () () () () () () () ()

Figure 30.2: The stock_detail table in the Server Explorer.

The resultant form has a SqlConnection and a SqlDataAdapter called SqlConnection1 and SqlDataAdapter1 respectively as shown in Figure 30.3.

	11111	2222			11	-	 11	2	2			1	 1	:			
Ug sqlConnection1	55×	plate	Adaş	ster	1												
G Design 🛛 HTML																	
														1	I		

Figure 30.3. The SqlConnection and SqlDataAdapter generated by Visual Studio.

2. Select the Generate Dataset menu option from the Data menu (you have to be in the design mode to see this option) or Select Generate DataSet from the property page of the SqlDataAdapter.

The Generate DataSet dialog box appears as shown in Figure 30.4. Call the DataSet dsStock and ensure that the box that says Add this DataSet to the designer is checked.

A Register M. Parcent Reveal (4 Mill Score)		alt.
the full time throad Build Debug De	a figwood Table Joanni Figunais Junio Spindow (198)	
13. 1. 0 9 9 1 4 B B	Cardinal a being a period	- 品は発明・〇、
(D) + (A	1 * / E AZ ==== E E E G G G
sevetatos 9 ×. El de 19 El	Battan motorian better and Matheman	· · · · · · · · · · · · · · · · · · ·
	Executed Education Energy of a statement of tables to be particulated. Concert descent P have P have	Alter Octation open Antimetric approximation Antimetrine Antimetric approximation Antimetric approximation
All Senar Captorer St. Control	General El Helle	L
March Darmon & D. C. A.	AND AND A COMBRENE	and the later land and an

Figure 30.4: Generating the DataSet.

 An instance of the DataSet called dsStock1 appears at the bottom of the form and the object dsStock.xsd displays in the Solution Explorer.

The DataGrid

Add a DataGrid to the web form, which will act as the main user interface. To do this, drag and drop a DataGrid from the Web Forms Toolbox onto the web form.

You must now set a number of properties (attributes) for the DataGrid. To do this, rightclick on the DataGrid and select Properties. The property page for the DataGrid will appear as shown in <u>Figure 30.5</u>.

Beautify the DataGrid by clicking on the AutoFormat hyperlink at the bottom of the property page and selecting Classic2. Now set the DataGrid properties as detailed below:

- DataSource= dsStock1
- DataMember = stock_master
- DataKeyField = code_value

These selections are shown in Figure 30.5 and are circled in red.

And the second statement and the second statement of the second	an a					_	ALL MADE IN		
the full time fromt Built Debug F	prist like in	int Fpr	ner Solt Street	0.0			Detailared's System. Not	NUE NABOLINE ON DATAS	ed at
J- J- 08 8 8 4 4 8 5	# E	9.6-th	a k 🗶 G	entik:					
1= 0 + (1						DataEndrap)		• E.(or)
index 0 m	Real Property lies	run pi	of her statements	March March		-11	(0)	CINNED	and a lot
ida .	Manual Academics		and the second second				Accessive		1012
inh Parma -							Also Catorinates	Table	1 21-
h mene							Man Page 10	rate .	- Prop
A Law							T desident with the		10.0
Ti lader							ApplementeColumn	Eus.	
							Seblier	White	pert. a
zj ezen	100000000000000000000000000000000000000						Sekiseptat		DL vefet
Towner		11111		11.1010			ander/der	#DEEFOG	
T notona		<u> </u>	to code redeated	Dist. and		160	Borther/Bulle	Roter	100
V WARNE		5.40	abo		abe	3	Bordenadth	Type:	- C
Dissionat					1.0	12.1	Carrieding		101.0p
rd later	1000000000	28	800	6.1	804		Cardonard	Tabana'	-Brister.
 Deterring 	1101010000	1余 2	abc	6.2	abs	9.2	Confilms	Disease.	17
Defeid		7.4. 3	44		100	14.1	Danke-Nill	Tide rate	freed of
E Papeatar	*********	- 444	40%		***		converter (stack_master	
P Chebber		LA 4	abe	6.4	104	2.4	Debelowce	z delitecki	
C Owdeniak	10000000.0						E0thadades		
E Rackebullarkal	1000003004						a tothedule		
# Ealabetter	Internet						Dubbi	True .	- 11
of lower	11010120120						Cubere-Cute	Due .	
- Frank							No. of Concession, Name		-1
Constanting .	11010000000						Address, feasts	0.68	
T Calendar									
TABASA									
314							AlexCustorsPaging		
- fan de fan							whether to hum on a pp	et for cultoricoging.	
A company and an	COLUMN 1								
Contraction of the second seco									
income is w	12 million	100	Contractores	10.000	dei .				
194									
Deboard ling	1 million (1997)								
leveral .	Geberge (E)	- 10							
Reads									

Figure 30.5: Setting the Properties of the DataGrid.

Now switch to HTML view and note that the DataSource, DataMembers, and DataKeyField attributes are added to the DataGrid element tag as follows:

<asp:datagrid id="DataGrid1"

DataSource="<%# dsStock %>"

DataMember="stock_master"

DataKeyField="code_value">

Selecting DataGrid Columns

If you do not already have the property page open, do so by right clicking (in the Design View) on the DataGrid and selecting Properties. Click on the hyperlink called Property Builder (this appears at the bottom of the Property page). Click on Columns. Select all the columns by clicking on All Fields in the Available Columns list box and then clicking the forward arrow (>). This is shown in Figure 30.6.

the star block through the				C. LEILA
On fig. Jaco Grines By	Al (phus Fornat Talà	a panel France Sub Mindow (1919		
1.3 . 1.0	B. C	The adverted to the Country	· 5.2273 ·	0.
1-	m. La lettine H		1 1	and any part of the state of the
	101 * 113 14 • F		1 1 # 7 1 10 2	
Teebox 8. H 9	Stack/Hawlet angin			Stantplay -C. # >
Defa E				
add form	Ste	ock Master Records		Solution: Chapter 37 (1 pro
# Plates	Add anew record.			= C+ Chapter 30
Aider	A REAL PROPERTY AND A REAL PROPERTY.			a girona
Li Tadên	DataGridt Property	tee		C Lundton
.#J 1/400	Eds 1 Closed	1		- Capterio estes
Tineus :	Closes.	F" Owder tokens automatically at rait time		8 A ordered
Inspire -	Edd 1 Millioners	Citize M		8 2 Meno
A MORER :	Lis 1 W. Pages	And de cherne Selec	ched exilence	
Distorest .	"A rown	at Distants + 136	88, updam, Cancel 👘 🖉	1 Deciminant
rd latter	And i Denies	Q (4 has) 30	ada	+ 9 Sudfate (
Deterring	Edg 2	C and dealer		- 39 televely
Disease .	20202	-0.40 el el	a set	
E rapage		Address of the second s		
9 Chebbox		Header Insti: Fuster	ried	555E (
C Owderial				
E Rackettoriat		Inches Practer	Concession of the second se	1000
# Eatellitter			a) Ø take	1111
104		10100		1000 T
C fani		ha ba	a sector	1111 C
C Patrick		part part		1000
T Calendar		Generation International Inter	N THE R	1000
TARIAN	9444	litera fred	anan 🖭	
144	Onen			1001
C feasifications				
P. Corportation				2
P. townshine				
Components	1510	Canal the column at a header. Column		
474		OK I	Canvel and I man 1	
Optional long	Anna Income	-		
Seres 19	a pendo list and			
Rendr			A	

Figure 30.6: Selecting the columns of the DataGrid.

If you view the form in the HTML view, you will note that the <Columns> element tag is generated as follows:

<Columns>

<asp:EditCommandColumn ButtonType="LinkButton"

UpdateText="Update" CancelText="Cancel" EditText="Edit">

</asp:EditCommandColumn>

<asp:BoundColumn DataField="code_value" HeaderText="code_value">

</asp:BoundColumn>

<asp:BoundColumn DataField="code_display" HeaderText="code_display">

</asp:BoundColumn>

<asp:BoundColumn DataField="rate" HeaderText="rate">

</asp:BoundColumn>

<asp:BoundColumn DataField="uom" HeaderText="uom">

</asp:BoundColumn>

<asp:BoundColumn DataField="closing" HeaderText="closing">

</asp:BoundColumn>

<asp:BoundColumn DataField="opening" HeaderText="opening">

</asp:BoundColumn>

</Columns>

Make the closing column read-only, as this column is updated automatically by a trigger on the stock_detail table. Also make the code_value column read-only, as this is the primary key column. To do this, in the HTML view, start typing, and Visual Studio.NET will try to auto-complete the tag by providing tag selections in a drop-down list. The closing and the code_value columns should look like this:

<asp:BoundColumn DataField="closing" HeaderText="closing" ReadOnly=True>

</asp:BoundColumn>

<asp:BoundColumn DataField="code_value" ReadOnly="True"

HeaderText="code_value">

</asp:BoundColumn>

Adding the Add, Edit, and Delete Hyperlinks

Now create the Edit, Update, and Cancel Button Columns (the Update and Cancel hyperlinks appear in the Update mode of the DataGrid). To do this, scroll down in the Available Columns box till you see Button Columns. Expand this list and you will see the option Edit, Update, Cancel as shown in <u>Figure 30.7</u>.



Figure 30.7: Selecting the Edit, Update, Cancel buttons.

Click the Add Button (the forward arrow). This will create an Edit hyperlink. If you view the web form in HTML view, you will note that the EditCommandColumn tags have been created with the UpdateText, the CancelText, and the EditText attributes. If you remember, these are the hyperlink captions that display with the appropriate update, cancel, or edit hyperlink. The Edit hyperlink displays initially in the DataGrid. When you click on it, the DataGrid displays in the Edit mode and the Update and Cancel hyperlinks display at this stage.

<asp:EditCommandColumn ButtonType="LinkButton"

UpdateText="Update"

CancelText="Cancel"

EditText="Edit">

</asp:EditCommandColumn>

Add a Delete Button in the same way as you added the Edit, Update, and Cancel Button Column. This Button Column selection will appear below the Edit, Update, and Cancel Button Column in the Available Column list. This is how the Delete ButtonColumn will look like in the HTML view:

<asp:ButtonColumn Text="Delete" CommandName="Delete">

</asp:ButtonColumn>

Now you have two hyperlinks in the DataGrid called Edit and Delete. If they are not beside each other, you can move them around using the up and down arrow, or remove them from the list by using the cross.

Drag and drop a LinkButton control from the ToolBox onto the web form. Place it over the DataGrid and under the page title. Right-click on it and select Properties. Set the following properties:

- ID = Add
- Text = Add a new record
- ToolTip = Add a new inventory master

Now, double-click on the LinkButton control. A blank method skeleton in the Code Behind file is created which is called <u>Add_Click</u>. We will code this method later. The skeleton created is as follows:

private void Add_Click(object sender, System.EventArgs e)

{

Convert Columns to Template Columns

Though the DataGrid is workable at this stage, it is better to convert the columns into Template columns. The most important advantage of doing this is that we can associate IDs with the EditItemTemplates that we can use to access the column values in the update mode. In the <u>Grid1_Update</u> method, we are required to extract the column values in order to build an update SQL query. In the absence of the IDs, we have to refer to the control using its index value as shown below:

TextBox t;

t = (TextBox)e.Item.Cells[2].Controls[0];

String code_display = " code_display = '" + t.Text.Trim() + "'," ;

For some reason, this does not always work as expected whereas using the ID of the control works reliably. As will be explained later, we can use the FindControl method of a control to locate a control of a specified ID within the DataGrid. This can be cast to a TextBox and its Text property extracted as shown below:

t = (TextBox) e.Item.FindControl("editCode_Display");

String code_display = " code_display = '" + t.Text.Trim() + "'," ;

To convert these columns to Template columns go back to the Property Builder and select Columns. Select code_display, rate, uom, and opening columns (each one separately) in the Selected Columns list box and click on the hyperlink that appears at the bottom which says Convert this column into a Template Column. Now, go to the HTML view and give unique IDs to the EditItemTemplates. As soon as you start typing, you will note that Visual Studio.NET will try to auto-complete the selection for you.

You will note that each Template column has an ItemTemplate and an associated EditItemTemplate. This EditItemTemplate can be any type of ASP.NET control (for example, a TextBox or a DropDownList control), whereas the ItemTemplate is a Label control. The ItemTemplate is the control that displays initially in the DataGrid. When you click on the Edit hyperlink in the DataGrid, the Edit- ItemTemplate displays. As this is an editable control like a TextBox, you can edit the displayed values.

Add the following IDs under the EditItemTemplate tag:

- The code_display column, the id will be editCode_display.
- The rate column, the id will be editRate.
- The uom column, the id will be editUom.
- The opening column, the id will be editOpening.

The <Columns> element tags should look like this in the HTML view:

<Columns>

<asp:EditCommandColumn ButtonType="LinkButton"

UpdateText="Update"

CancelText="Cancel"

EditText="Edit">

</asp:EditCommandColumn>

<asp:BoundColumn DataField="code_value"

SortExpression="code_value" ReadOnly="True"

HeaderText="code_value">

</asp:BoundColumn>

<asp:TemplateColumn HeaderText="code_display">

}

<ItemTemplate>

<asp:Label runat="server" Text='<%# DataBinder.Eval

(Container, "DataItem.code_display") %>'></asp:Label>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox ID="editCode_Display" runat="server"

Text='<%# DataBinder.Eval(Container, "DataItem.code_display")

%>'></asp:TextBox>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="rate">

<ltemTemplate>

<asp:Label runat="server" Text='<%# DataBinder.Eval

(Container, "DataItem.rate") %>'></asp:Label>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox id="editRate" runat="server"

Text='<%# DataBinder.Eval(Container, "DataItem.rate") %>'></asp:TextBox>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:TemplateColumn HeaderText="uom">

<ltemTemplate>

<asp:Label runat="server" Text='<%# DataBinder.Eval

(Container, "DataItem.uom") %>'></asp:Label>

</ItemTemplate>

<EditItemTemplate>

<asp:TextBox id="editUom" runat="server"

Text='<%# DataBinder.Eval(Container, "DataItem.uom") %>'></asp:TextBox>

</EditItemTemplate>

</asp:TemplateColumn>

<asp:BoundColumn DataField="closing"

ReadOnly="True" HeaderText="closing">

</asp:BoundColumn>

<asp:TemplateColumn HeaderText="opening">

<ItemTemplate>

<asp:Label runat="server" Text='<%# DataBinder.Eval

(Container, "DataItem.opening") %>'></asp:Label>

</ltemTemplate>

<EditItemTemplate>

<asp:TextBox ID="editOpening" runat="server"

Text='<%# DataBinder.Eval(Container, "DataItem.opening") %>'>

</asp:TextBox>

</EditItemTemplate>

</asp:TemplateColumn>

</Columns>

If you run the web form at this time, you will note that each column appears twice in the DataGrid. This is because the DataGrid is auto-generating columns, in addition to the Template columns we have created. You can do this either by setting the

AutoGenerateColumns property of the DataGrid to False or by unchecking the Create columns automatically at run time check box in the Property Builder as shown in Figure 30.8.

A Destination of the second second			1.111
De Lift See Don't	Bull Drive Fernat Tak	in Junet France Solle Window (194)	
D D So	L D. R	E state state s 2.00 Carde B .	0.
	and a local state of the		in a second second second
	= 10 = [] H		in a self in the set .
feebox 8 ×	Stack/faster.aspx		CELE Selder Egiere - C., # 1
2da			
and Parms -	St	ock Master Records	Solution (Degiter 37 () pro
 Platter 	L. Addance record		= C+Chapter-30
ALder	·		a green
il laden	Detected Proce	rtsa 🖸	C Lundton a
#] 1.400	Eds 1 Cloved		- Capterio edes
TI neeroe	There are	Conta stars admittative reventee	8 A ordered
T motores	2.6.6	Char N	a a conductor
V HOREN	Lig 1 St. Pages	Ander cheve: Second care	- A term to
Designation 1	A row	D Data Partis A 2008, Updates, Cavall A 7	0 Distributer age
rd later	t and a monthles	Contract Science	+ E Stadhate A
Denirel	- La 1	Cash Apple	- 2 velocity
Distant Co.	entropy -	X k k k k k	12222
E rapearer	100000	Million and Manual States	10000 C
F Chebber	10000	Header Texts: Fuster texts	
C OwdersM	627223		12222
E Rabibilion Ref.	ACC 810	Header Page: East expression:	1000
# Tabilitie	10000	- y Vinke	111111
and through	100001	Elliptic Covelant	1000
Finit	1.11.11.11.11 1.11.11.11	Est Eanut	1000 C
 Harmony 	10000	Ladule Insti-	11111
Calerdia	111111	pere britation #	10000
- ARidata	- Teneral		10000
14k	1.000		12221-
C fesantinitian	1		
P3 Corportaidator	<u></u>		10
P. torombian	10-11	Convert the relation at a baseline relation	
with a	24		
Onivertilan		OK Caved And Take	
(men)	Ge design TAL HERE		
Reals		the second se	
strend in the second	10.01.00 - 1 - mm	the state of the second s	C1.18.0

Figure 30.8: Setting the AutoGenerateColumn attribute to False.

If you look at the generated HTML in the HTML view, you will note that the attribute AutoGenerateColumns="False" has been inserted within the DataGrid element tag as follows:

<asp:datagrid id="DataGrid1"

AutoGenerateColumns="False">

The Add Panel

The functionality to insert new rows to the stock_master table is provided by a number of TextBox controls and Label controls residing on a Panel control. The reason why we have these controls on a Panel control is that we can show or hide all the TextBox controls and Label controls just by setting the Visible property of the Panel to True or False.

Drag and drop a Panel control from the ToolBox onto the web form. Right-click on the Panel control and select Properties. Give it an ID of AddPanel and set the Visible property to False.

Now, drag and drop four Label controls and four TextBox controls on the Panel. These should line up one under the other. To line them within the Panel, drag and drop the first TextBox and Label. Then right-click and copy the label and paste it one line below. Stretch the second label with your mouse until it lines up with the Label on the top. Now copy and paste the TextBox so that it sits next to the label on the second line. Repeat this process for all the Label and TextBox controls you need on the Panel. You need to

follow this process in order to line up the components. The TextBox controls will be of similar size, however the Label sizes can vary, depending on the caption you want to display, hence you need to stretch the Label controls so that they line up under each other. You can use your mouse or CRTL-arrows to move the controls and SHIFT-arrows to size them.

Give the four TextBox controls IDs of AddCode_Display, AddRate, AddUom, and AddOpening respectively. You can do this by right clicking on each of them and selecting Properties and then ID.

Add a Button Control onto the Panel. Give it the ID of SaveNew. Double-click on it to create a method skeleton, which will look like this:

private void SaveNew_Click(object sender, System.EventArgs e)

{

}

We will code this method later. I have also added a blank label before the Button Control in order to position it in the center of the form.

Specifying the DataGrid Command Methods

Specify the OnEditCommand, OnCancelCommand, OnUpdateCommand, and the OnDeleteCommand within the DataGrid element tag. These attributes of the DataGrid specify the method that will fire when the user clicks on the Edit, Cancel, Update, or Delete hyperlinks in the DataGrid. You will have to open the form in HTML view and manually add these attributes within the DataGrid element tag as follows:

<asp:datagrid id="DataGrid1"

------OnEditCommand="Grid1_Edit" OnCancelCommand="Grid1_Cancel" OnUpdateCommand="Grid1_Update" OnDeleteCommand="Grid1_Delete" >

Methods

In this step, you will code some methods for the web form. Open the Code Behind file (StockMaster.aspx.cs) and add the following methods.

The Bind() method will populate the DataSet using the Fill method of the DataAdapter. It will also bind the DataGrid to the DataSet as follows:

The Bind Method

public void Bind()

{

sqlDataAdapter1.Fill(dsStock1);

DataGrid1.DataSource=dsStock1;

DataGrid1.DataBind();

if (AddPanel.Visible==true)

{AddPanel.Visible=false;}

}
The Bind method is called in the Page_Load event as follows:

private void Page_Load(object sender, System.EventArgs e)

{

if (! IsPostBack)

{

Bind();

}

If you build and run the web form (by pressing CTRL-F5), you will see the form displayed in the browser as shown in Figure 30.9.

	nis @hearbonal Clouge 3	-(Sani	1. 1910	attal.	Otan without a Ca	utoplan @recon	1 6.00
	Stock Master Re	core	ds				-
10.00	also rate display		œ	(SIN)	Contraction of the local division of the loc		
Ede 1	Lur Soap	8	dot	10	0		
Edt 2	Johnson & Johnson Scop	9.5	dog	0	0		
Eds 3	Coconst Od	100	Tee	0	0		
Edit 4	ASPORT	0	915	0	0		
<u>Edit</u> 7	int	0	212	-10	130		

Figure 30.9: The Stock Master form displaying data.

The <u>Gridl_Edit</u> method fires when the user clicks on the Edit LinkButton. It sets the EditItemIndex of the DataGrid to the clicked row and calls the Bind method as follows:

Grid1_Edit

public void Grid1_Edit(Object sender, DataGridCommandEventArgs e)

{

DataGrid1.EditItemIndex = e.Item.ItemIndex;

Bind();

The <u>Grid1_Cance1</u> event fires when the user clicks on the Cancel LinkButton in the Edit mode of the DataGrid. This method simply sets the EditItemIndex to -1, which in effect tells the DataGrid that no row is now selected and that it should close the update mode. The Bind method is then called to refresh the DataGrid.



public void Grid1_Cancel(Object sender, DataGridCommandEventArgs e)

{

DataGrid1.EditItemIndex = -1;

Bind();

}

The <u>Grid1_Delete</u> event fires when the user clicks on the Delete LinkButton in the DataGrid. This event extracts the primary key of the clicked row and builds a SQL Delete query, which it hands over to the method <u>RunSql</u> for execution. Finally it sets the EditItemIndex to -1 and refreshes the DataGrid by calling the Bind method.

Grid1_Delete

public void Grid1_Delete(Object sender, DataGridCommandEventArgs e)

{

int Key=(int)DataGrid1.DataKeys[(int)e.Item.ItemIndex];

String code_value = Key.ToString() ;

String s = "Delete from stock_master";

s +=" where code_value = " + code_value;

RunSql(s);

DataGrid1.EditItemIndex = -1;

Bind();

The <u>Gridl_Update</u> method is fired when the user is satisfied with the changes he has made to a record in the Update mode and clicks on the Update LinkButton. The method

extracts the primary key by looking at the DataKey property of the clicked row. Then it uses the FindControl method of a control to locate its ID within the DataGrid. The returned object is cast to a TextBox and its Text property extracted as shown below. A SQL update query is built using the extracted Text properties of the TextBox controls, which is then handed over to the RunSql method for the actual execution. Finally the EditItemIndex is set to -1 and the Bind method called to refresh the DataGrid. Grid1 Update

public void Grid1_Update(Object sender, DataGridCommandEventArgs e)

{

TextBox t;

//This is the primary key

int Key=(int)DataGrid1.DataKeys[(int)e.Item.ItemIndex];

```
String code_value = " code_value =" + Key.ToString() ;
```

t = (TextBox) e.Item.FindControl("editCode_Display");

```
String code_display = " code_display = '" + t.Text.Trim() + "'," ;
```

t = (TextBox) e.Item.FindControl("editRate");

```
String rate = " rate = " + t.Text.Trim() + "," ;
```

t = (TextBox) e.Item.FindControl("editUom");

```
String uom = " uom ='" + t.Text.Trim() + "'," ;
```

t = (TextBox) e.Item.FindControl("editOpening"); String opening = " opening = " + t.Text.Trim() ;

String s = " Update stock_master Set";
s += code_display +rate +uom ;
s += opening ;
s += " Where " + code_value;

RunSql(s);

DataGrid1.EditItemIndex = -1;

Bind();

```
This method will be used to execute SQL insert, update, and delete commands.
```

public String RunSql(string vsql)

```
{
try
{
    SqlCommand mycommand = new SqlCommand(vsql,sqlConnection1);
    sqlConnection1.Open();
    mycommand.ExecuteNonQuery();
    sqlConnection1.Close();
}
catch(Exception e)
{
    string ret = "Exception: " + e.ToString() ;
    messages.Text=ret;
}
```

return("OK");

The <u>Add_Click</u> event fires when the user clicks on the Add LinkButton. This event initializes the TextBox controls with default values and displays them by setting the Visible property of the AddPanel to True. Add_Click private void Add_Click(object sender, System.EventArgs e)

{

AddPanel.Visible=true;

AddCode_Display.Text= "";

AddRate.Text= "0";

AddUom.Text= "";

AddClosing.Text= "0";

The <u>SaveNew_Click</u> event fires when the user clicks on the <u>SaveNew</u> button that resides on the <u>AddPanel</u>. This event first sets a default value to any TextBox control not filled in by the user. It then builds a SQL Execute query call to the stored procedure p_stock_master and passes it the required parameters by extracting the <u>Text</u> properties of the various TextBox controls. This procedure was discussed in <u>Chapter 24</u> (Inventory Masters) and if you remember, to insert a new record, you pass it a <u>NULL</u> code_value. This Execute query is handed over to the <u>RunSql</u> method that handles the actual execution. Finally the EditItemIndex is set to -1 and the Bind method is called to refresh the DataGrid.

```
SaveNew_Click
```

private void SaveNew_Click(object sender, System.EventArgs e)

{

```
if (AddCode_Display.Text.Length == 0)
```

{ messages.Text="Sorry - Account Name cannot be blank";

return;

}

```
if (AddRate.Text.Length == 0)
```

```
{AddRate.Text= "0";}
```

```
if (AddUom.Text.Length == 0)
```

```
{AddUom.Text= "";}
```

```
if (AddClosing.Text.Length == 0)
```

{AddClosing.Text= "0";}

String s = "Execute p_stock_master ";

s += " @code_value=NULL," ;

s += " @code_display = '" + AddCode_Display.Text + "',";

s += " @ rate = " + AddRate.Text + ",";

s += " @uom = '" + AddUom.Text + "',";

```
s += " @closing = " + AddClosing.Text ;
```

messages.Text= "";

RunSql(s);

DataGrid1.EditItemIndex = -1;

Bind();

This concludes our discussion on using Visual Studio.NET to build CRUD applications. No doubt you might have noticed the significant advantages of scripting using this tool. I must admit that I am quite impressed with the auto-completion features of Visual Studio.NET that work with the element tags in HTML view. I no longer need to remember the various attributes associated with a tag as I can see them displayed as soon as I start typing. The ability to move (and place) the controls around with my mouse is another feature that I like.

Chapter 31: Creating a Web Service Using Visual Studio.NET

In this chapter, I show you how to develop and consume a web service using Visual Studio.NET. You will first build a generic database service in C#. The web service will have a method called Populate, which will accept a SQL SELECT query. Based on this SELECT query, the web service will return a DataSet to the calling object that will contain the result set of the SELECT query. This web service will have another method called RunSQL, which will be used to apply an action query (that is, an insert, an update, or a delete query) to the database.

You will then build another Web project that will *consume* this service. This project will call the Populate function of the web service with appropriate parameters and receive the DataSet that contains the result set. A DataGrid will then be bound to this DataSet to display the rows returned from the database. You will also test out the RunSQL method by inserting and deleting a few rows.

Building the Generic Database Web Service

Here is how you will build the database web service:

- 1. Start Visual Studio.NET.
- 2. Select File/New/Project.
- 3. Select Visual C# Projects from the left pane and ASP.NET web services from the right pane. Figure 31.1 shows VS.NET at this stage. (In the Release Candidate version of Visual Studio.NET one can append the project name—CsharpdbService at Location. i.e.

http://localhost/CSharpdbService. The Name field is disabled and will automatically display the project name one typed in.)

tieboi # Deta IH Schena	A ServiceLansacci(Design)		10.		5
te Factor	New Project		ET al	E Ghapition E Ghapition - Ghapition	
	Head Tree House Tree The Tree			S Charada B Charada S Security S Televerity	i dan sedan K
	A project for evening with our axes to use the	a ate sydenee			
	Nese Chapteries Locator: Vip.(Real-of	2	Poet.		
	Chalchelinken /F Gase Jaka	b m		for cash anno 7517	- contract
Components weature terms	* Project reliae present of Http://docab.oc/City	rodiferatu.		15.00	
Calceditra	Pag	OK Canol	**	Build Artists	Cattern
Owput				Custon Tod Nameson Pin Name Full rate	Service (service)
				Rufel Action Monit's Technologie (precesses)	he bolit and
Tions -	internet in the second s			1	1.

Figure 31.1: Creating a new C# ASP.NET web service.

- Type in CSharpdbService for the name.
 Visual Studio.NET generates a new solution, which creates a reference folder and four files. The Web.config is an XML file, which contains various configuration options (for example, session timeout interval), and which controls the web service at runtime. The file CSharpdbService.vsdisco is an XML file used for dynamic discovery of web services by clients. This means that when you want to use this service in another project, you navigate to this file and VS.NET adds a reference to the service for you. The file Global.asax is where projectwide event handlers (such as ApplicationStart and ApplicationEnd) reside.
- 6. Right-click on Service1.asmx and select Open With and then Source Code(Text) Editor (or navigate to the application directory and open the file in Notepad). You will note that this file makes a callout to code that resides in another file (Service1.asmx.cs) as follows:
 - <%@ WebService Language="c#" 7.

Codebehind="Service1.asmx.cs" Class="CSharpdbService.Service1" %>

At this stage, you have all the pieces of the web service in place and you just need to code the web services (asmx) file.

Scripting the Web Service

In this section, I will create the web service called dbService. This web service will have two methods, Populate and RunSql, as described in the following steps.

- 1. Delete the default Service1.asmx file and add a new web service file by right-clicking the project name and choosing Add/Add web service. You can call this web service dbService.asmx.
- 2. A web service template form is created which includes several namespaces. The following namespaces are included in dbService.asmx.cs:
 - 3. using System;

- 4. using System.Collections;
- 5. using System.ComponentModel;
- 6. using System.Data;
- 7. using System.Diagnostics;
- 8. using System.Web;

using System.Web.Services;

Because you need to interact with databases, you will import the namespaces for either the SQL or OleDb Managed Provider. I have chosen to work with the OleDb Managed Provider because I want to keep the dbService web service as generic as possible. Using SQL Managed Provider would limit its use to Microsoft SQL Server databases. You will need to import the OleDb Managed Provider namespace as follows:

using System.Data.OleDb;

The web service is created in the namespace CSharpdbService as shown:

namespace CSharpdbService

{
 //all the code goes here

}

The class is called dbService and it inherits from System.Web.Services.WebService. The inheritance is indicated by the use of the colon (:) before the base class, as shown in the following line:

public class dbService : System.Web.Services.WebService dbService has a field called connStr, which you must declare with a private scope as follows:

private String connStr;

The dbService class has a constructor that has no parameters. A constructor is a method that has the same name as the class. Each class has a default constructor, which does not accept any parameters. A class is free to add more constructors that may accept parameters. In the constructor, the connection string is set to the private field connStr as follows:

public dbService()

{

InitializeComponent();

connStr = "Provider=SQLOLEDB; Data Source=(local); ";

connStr = connStr+" Initial Catalog=PUBS;User ID=sa;Password=";

9. Two Web Methods need to be created within the body of the dbService class. A Web Method is created like a normal C# method; however, it is preceded by the attribute [WebMethod].

The first method is called Populate and it is used to send a SELECT SQL query to the database and receive the results back in a DataSet. This DataSet can then be used to bind a bound control like the DataGrid. Here is the listing of this method:

[WebMethod]

public DataSet Populate(string SQL)

{

//for queries that return data

//and for binding controls

OleDbConnection myConnection = new OleDbConnection(connStr);

OleDbDataAdapter myCommand = new OleDbDataAdapter(SQL, myConnection);

```
DataSet ds = new DataSet();
```

myCommand.Fill(ds, "vTable");

return ds;

}

- The second method is called RunSQL, which is used to apply an action query to the database. An action query, as you know, is a SQL query, such as insert, delete, or update, which performs an action against a database and does not return any data. This method accepts a valid SQL insert, delete, or update query and applies it to the database. Here is the code for this method:
 - [WebMethod]
 - public String RunSQL(string vsql)
 - {
 - try
 - {
 - OleDbConnection myConnection = new OleDbConnection(connStr);
 - OleDbCommand mycommand = new OleDbCommand(vsql,myConnection);
 - myConnection.Open();
 - mycommand.ExecuteNonQuery();
 - myConnection.Close();
 - return("OK");
 - -
 - }
 - catch(Exception e)
 - {
 - string ret = "Exception: " + e.ToString() ;
 - return ret;
 - }

}

Your web service is now ready. You can test it by right-clicking the web service asmx file (dbService.asmx) in the Solution Explorer and selecting Set As Start Page. Now press F5 to build and display the web service test page as shown in Figure 31.2.



Figure 31.2: The test page for the web service.

Now, click on the method Populate. In the page that appears provide the parameter value:

Select * from authors

When you click on the invoke button, all the records from the authors table are displayed in XML format as shown in <u>Figure 31.3</u>.



Figure 31.3: The Populate method returns database rows as XML.

Here is the complete code listing of the web service: **dbService.asmx**

using System;

- using System.Collections;
- using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Web;

```
using System.Web.Services;
```

```
using System.Data.OleDb;
```

```
namespace CSharpdbService
```

{

```
/// <summary>
```

```
/// Summary description for dbService.
```

```
/// </summary>
```

```
public class dbService : System.Web.Services.WebService
```

{

```
public dbService()
```

{

```
//CODEGEN: This call is required by the ASP.NET Web Services Designer
```

```
InitializeComponent();
```

```
connStr = "Provider=SQLOLEDB; Data Source=(local); ";
```

```
connStr = connStr+" Initial Catalog=PUBS;User ID=sa;";
```

```
}
```

```
private String connStr;
```

```
#region Component Designer generated code
```

```
/// <summary>
```

```
/// Required method for Designer support - do not modify
```

```
/\!/\!/ the contents of this method with the code editor.
```

```
/// </summary>
```

```
private void InitializeComponent()
```

```
{
```

```
}
```

#endregion

```
/// <summary>
```

```
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
}
[WebMethod]
public DataSet Populate(string SQL)
{
 //for queries that return data
 //and for binding controls
 OleDbConnection myConnection = new OleDbConnection(connStr);
 OleDbDataAdapter myCommand = new OleDbDataAdapter(SQL, myConnection);
 DataSet ds = new DataSet();
 myCommand.Fill(ds, "vTable");
 return ds;
}
[WebMethod]
public String RunSQL( string vsql)
{
 try
 {
  OleDbConnection myConnection = new OleDbConnection(connStr);
  OleDbCommand mycommand = new OleDbCommand(vsql, myConnection);
  myConnection.Open();
  mycommand.ExecuteNonQuery();
  myConnection.Close();
  return("OK");
```

```
catch(Exception e)
{
  string ret ="Exception: " + e.ToString() ;
  return ret;
}
}
}
```

Calling the Web Service from a Web Form

In this section, I will build a client application, which will use the web service developed in the preceding section. This application is a C# ASP.NET Web Application and is created as explained in the following steps.

- 1. Start Visual Studio.NET. and open the CSharpdbService Solution created earlier (this Solution needs to be open because you will add the new project to it).
- 2. Select File/New/Project.
- 3. Select Visual C# Projects from the left pane and ASP.NET Web Application from the right pane. Select the option button that says Add to Solution, as shown in <u>Figure 31.4</u>. Doing so will add this new project to the current solution and you can work with both the projects from within the same solution. Type in **TheClient** for the project name. In the Release Candidate version of Visual Studio.NET one can append the project name—TheClient at Location. i.e. http://localhost/TheClient. The Name field is disabled and will automatically display the project name one typed in.

art		New Product			
	-	Pergentil Troome	Templaters	[HT III]	
Cert Usartad actor's New Delate Constructly Headfree Described Described Described Actor Rocking actor Rocking actored in	VSOvers CSharpd Bundom Teerage Oper	Visal Tax Frents Visal Criments Visal Criments Visal Criments Visal Criments Visal Rode Solars		gge a ' consultany] gge a ' vak const they _	
		Name: TerOwn			
		Location: Very United and Additional Control of the Sound of Here Control of the Sound of Here Control on the Control of the Sound of Here Control on the Sound o	en Frank	Bitate	
			<u></u>		

Figure 31.4: Creating a new C# ASP.NET Web Application.

You will see both the projects in the Solution Explorer, as shown in Figure 31.5.



Figure 31.5: The two projects displayed in the Solution Explorer.

- 4. Right-click on the aspx form called WebForm1.aspx, which was created by default, and mark it as the startup page by selecting Set as Start Page. You also need to mark the project TheClient as the startup project. To do this right-click on the project name in the Solution Explorer and select Set as StartUp Project.
- 5. Right-click on an empty portion of the form and select Properties. Make sure that the Page Layout property is GridLayout. With GridLayout, you are able to drag the controls and position them visually on the form. Drag one DropDownList control, two Label controls, one DataGrid control, and two Button controls onto the form. Right-click on each of the buttons to bring up their Properties page and give them IDs of Populate and RunSql. The first label control is used to give a descriptive title to the form. The second label is given an ID of messages. Bring up the property pages for the DropDownList control (right-click on it and select Properties). Locate the items property and click on the three dots (ellipsis) beside it to bring up the ListItem Collection Editor. Click on the Add button located at the bottom of this panel. You need to add a few SELECT SQL queries in the Value input box. For example, you can enter the following queries:
 - 6. Select * from authors
 - 7. Select * from titles
 - 8. Select * from publishers

Select * from pub_info where pub_id = "9999"

The user will select one of these SELECT query statements from the dropdown list, which will be passed to the <code>Populate</code> method of the web service. The web service will return a DataSet, which will be used to bind the DataGrid.

I have beautified the Web page by right-clicking on an empty portion of the form and selecting Properties. This brings up the Document Property Pages window, from where I selected the Color and Margins tab. I clicked on the three dots (ellipsis) appearing next to the BackGround color box and visually selected a background color for the form. This was the color #ffcc99.
Next I worked on beautifying the DataGrid. I right-clicked on the DataGrid to bring up its property page. I clicked on the hyperlink AutoFormat located at the bottom of the property page. I selected the Classic2 color scheme from the presented list of styles.

This web form now looked like the one displayed in Figure 31.6.

Testing the Web Service:						
Select * here authors	Test Populate	Test Radia				
Columni	Celumal	Column				
all c	abe	abc				
alte	der.	also .				
alto	abe:	abe				
atc	abe	abc.				
atc	des .	des				
Corp. (3+14						

Figure 31.6: Design of the client web form.

9. You need to add a reference to the CSharpdbService to use it in this application. To do this, select Project/Add Web Reference from the main menu or right-click on the project name in the Solution window and select Add Web Reference.

A new window called Add Web Reference appears. On the left side of the window click on the hyperlink Web Reference on Local Web Server. This should show you all the discovery files (*.disco or *.vsdisco) from which you should click on the link CSharpdbService.vsdisco. Visual Studio.NET will discover the web service and display results, as shown in <u>Figure 31.7</u>. You can also type in the fully qualified (localhost) address of the web service, suffixed by a ?wsdl in the Address box. For example on my machine, the web service resides at



http://localhost/CSharpdbService/dbService.asmx?wsdl.

Figure 31.7: Web service discovery.

Now click on the Add Reference button at the bottom right of the window. A reference to the web service is added in the Solutions Explorer as shown in Figure 31.8.



Figure 31.8: Reference to web service in the Solutions Explorer.

10. Now add a few lines of code to make it all work. First declare and instantiate the web service as follows:

private localhost.dbService mydbService = new localhost.dbService();

- Now code the click event of Button1. Please note that the click events of the two buttons must appear after the InitializeComponent() method as the delegates for the click events for these buttons are created in this method. If you place these events before the InitializeComponent() method, they will not fire.
 - private void Populate_Click(object sender, System.EventArgs e)
 - {
 - DataSet ds = new DataSet();
 - String s = DropDownList1.SelectedItem.ToString();
 - ds = mydbService.Populate(s);
 - DataGrid1.DataSource=ds;
 - DataGrid1.DataBind();

}

- This tests the Populate method. The user-selected SELECT query is sent to the Populate method of the web service. A DataSet is returned which is used to bind the DataGrid.
- The web form after you press F5 and test the Populate method should look like the screenshot in Figure 31.9.
- To Test the RunSQL method of the web service, add the following code behind the click event of the RunSql button.
 - private void RunSql_Click(object sender, System.EventArgs e)
 - {

- string s;
- String ret;
- s = "Delete from pub_info where pub_id = '9999'";
- ret=mydbService.RunSQL(s);
- messages.Text= ret;
- •
- //insert a random number
- Random r = new Random();
- •
- s = " insert into pub_info(pub_id,pr_info) values(";
- s = s + " '9999','" + r.Next(1000).ToString() + "')";
- ret = mydbService.RunSQL(s);
- messages.Text= ret;
- •
- //Refresh the DataGrid to show changes
- DataSet ds = new DataSet();
- s = "Select * from pub_info where pub_id = '9999'";
- ds = mydbService.Populate(s);
- DataGrid1.DataSource=ds;
- DataGrid1.DataBind();

•	
	3

(Cone	erine Char	nenal (C.ior	a Asan	nini Ohan webigi	ca ękuloj	dare d	() Helcone		
Testing	the Web	Service	:						ŕ
Galant Theor	authors		2	Test Populate	Test Rust	W.			
m,id	an joanne	en france	phone	aldress.	city	022	10	(matrix)	
172-32- 1176	Whate	Johnson	400-496- 7228	19932 Bigge R.4.	Merio Park	CA.	\$4025	3rae	
113-46- 1915	Crea	Majorit	415 985- 7620	309 63rd St. #411	Oakland	C.A	54618	2ve	
238-95-	Carson	Cheryl	415 548- 7723	589 Darwin La.	Berloeky	CA	\$4705	210	-
207-41-	OLeary	Michael	408.285- 2428	22 Cleveland Av. #14	San Jose	CA	95128	Ine	
274-38-	Strage	Dean	415.834- 2919	5420 Cullege Ax	Ouldand	CA.	\$4609	2ve	
M1-22-	Suit	Measter	913 043- 0462	10 Marsinippi Dr.	Lawrence	ĸs	65044	False	
09-56- NOB	Depart	Abraham	415 658- 9932	6223 B stream 94	Detorky	CA	54705	2ne	
(27-17- 1319	Del	Ann	415 £36- 7128	3410 Bloode 3t.	Pais Alto	CA	94301	214	
172-27-	Calutation		767.538-	20.2-200	And a	~		-	

THE ALL OF ALL DE THE ALL DE THE SOLUTION ALL DE SOLUTION SOLUTIAN SOLUTIA SOLUTI

- Figure 31.9: Selection of a SELECT query displays the results in a DataGrid.
- This method uses the RunSql web service method to first delete rows from the pub_info table having the pub_id of 9999. It then uses the Random class to generate a number between 1 and 1000. Note that the Next method of the Random class does this, based on a parameter passed to it. This parameter specifies the upper limit of the random number, which I have specified to be 1000. This number is inserted into the pr_info column of the pub_info table, for the pub_id of 9999. Finally, the Populate method is called to return the changed rows. The DataGrid is again bound to the returned DataSet, thus showing the most current values.

You can test this method by clicking on the Test RunSql button. Each time you
perform this action, you will see a new number in the pr_info column, as
shown in Figure 31.10.



Figure 31.10: The Test RunSql button inserts a random number in the table.

Here is the complete listing for the Code Behind file WebForm1.aspx.cs. **WebForm1.aspx.cs**

using System;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Web;

using System.Web.SessionState;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Web.UI.HtmlControls;

namespace TheClient

{

/// <summary>

```
/// Summary description for WebForm1.
```

```
/// </summary>
```

{

```
public class WebForm1 : System.Web.UI.Page
```

```
protected System.Web.UI.WebControls.DataGrid DataGrid1;
protected System.Web.UI.WebControls.Label Label1;
protected System.Web.UI.WebControls.DropDownList DropDownList1;
protected System.Web.UI.WebControls.Button RunSql;
protected System.Web.UI.WebControls.Button Populate;
private localhost.dbService mydbService = new localhost.dbService();
public WebForm1()
{
 Page.Init += new System.EventHandler(Page_Init);
}
private void Page_Load(object sender, System.EventArgs e)
{
// Put user code to initialize the page here
}
private void Page_Init(object sender, EventArgs e)
{
 //
 // CODEGEN: This call is required by the ASP.NET Web Form Designer.
 //
 InitializeComponent();
}
#region Web Form Designer generated code
```

/// <summary>

```
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.Populate.Click += new System.EventHandler(this.Populate_Click);
    this.RunSql.Click += new System.EventHandler(this.RunSql_Click);
    this.Load += new System.EventHandler(this.Page_Load);
```

}

```
#endregion
```

```
private void Populate_Click(object sender, System.EventArgs e)
{
```

```
DataSet ds = new DataSet();
```

String s = DropDownList1.SelectedItem.ToString();

```
ds = mydbService.Populate(s);
```

DataGrid1.DataSource=ds;

```
DataGrid1.DataBind();
```

}

```
private void RunSql_Click(object sender, System.EventArgs e)
```

{

```
string s;
```

String ret;

s = "Delete from pub_info where pub_id = '9999'";

ret=mydbService.RunSQL(s);

messages.Text = ret;

//insert a random number

Random r = new Random();

s = " insert into pub_info(pub_id,pr_info) values(";

s = s + " '9999','" + r.Next(1000).ToString() + "')";

ret=mydbService.RunSQL(s);

messages.Text = ret;

//Refresh the DataGrid to show changes

DataSet ds = new DataSet();

s = "Select * from pub_info where pub_id = '9999'";

ds = mydbService.Populate(s);

DataGrid1.DataSource=ds;

DataGrid1.DataBind();

}

}

Creating web services with Visual Studio.NET is quite simple. The provided tools and wizards make the task of laying out components on a web form a breeze. The automatic code generated by the tool is compact and concise and I really like this aspect. A major problem with earlier Microsoft tools like Visual InterDev was that they generated tons of code in supplementary files. This is not so with Visual Studio.NET. You can still open the generated files in a text editor and continue working.

Project 5 Summary

In this part, you saw the Rapid Application Development (RAD) capabilities of Visual Studio.NET. You experimented with the various tools, designers, and components of this environment and also developed a database and a Web services application using this tool.

As a RAD tool, Visual Studio.NET handles well. Its integrated development environment enables you to work with multiple languages and comes with an excellent debugger. Visual Studio.NET draws its graphical user interface from earlier Microsoft tools like Visual InterDev and Visual Basic 6x. A major problem with ASP development using Visual InterDev was that it generated quite a bit of code behind the scenes. Experienced programmers shied away from using it and preferred to use a text editor instead. You will be pleased to note that this is not the case with Visual Studio.NET. The code it generates is elegant and concise and in most cases quite similar to what you would code by hand. You can use the same tool to script in languages like C#, VB.NET, and ASP.NET, hence you have a low learning curve. All in all, I think that this is a welldesigned tool which can lead to enhanced developer productivity.

Part III: Appendixes

Appendix List

Appendix A: Installing the Sample Database Appendix B: HailStorm

Appendix A: Installing the Sample Database

The samples in this book require you to have an MS SQL Server database called ASPNET. Please complete the following steps to create the required database objects.

- 1. Open MS SQL Query Analyzer (isql). Do not change the default database (the master database).
- 2. In isql select File/open. Browse to the ..samples\database folder and select the file called Create.sql.
- 3. Execute this file by clicking the green arrow or pressing F5.

Figure A.1 shows create.sql loaded in the isql utility.

A LE CR. OR J. HIGH		A00.2
"This file monoming weriging for decidence objects to second, by the book (0) .NTF forestional Projects 17 Hores Gharin 19 Hores Gharin 19 Hores Gharin 19 Hores Gharin 19 Hores Gharin 19 Hores Gharin 10		-
tepl i Ecocute this file is the NUL Berver Query Analyser		
9		
reate Deciminate		
tit mastar		
0		
IF IN INCLUSION IN THE NULL BRACK		
DEATE DATABULE ANDRET		
THE ADDRET		
19		
CARLE CREATION		
MEATE TAILS groups		
(id that (b) NUL,		
code_valas integar 307 Mill.		
4		
Accessful Rooted away the STARPHETGraphini Exclusion prodelucial	East times 0x88.00 Extens	LAL, CHI
	Convertions 1	

Figure A.1: create.sql is shown loaded in the isql utility.

List of Database Objects Created

After running Create.sql, various database objects are created. The list of the tables, stored procedures, and triggers created are given below.

Database Objects Required for Financial Accounting

These objects are used by almost all of the examples developed in this book. <u>Project 1</u> (A Personal Finance Manager) in particular uses all of these objects.

Tables

Here is a list of the tables used for financial accounting.

- groups
- masters
- tr_header
- transactions
- tblSelection

Triggers

The transactions table has an update, delete, and insert trigger. These triggers update the closing balance field of the masters table as and when a financial transaction is created, deleted, or updated.

- delete_mstr: Delete trigger on transactions
- insert_mstr: Insert trigger on transactions
- update_mstr: Update trigger on transactions

Stored Procedures

Two stored procedures are used by the financial system. p_masters is used to insert or modify a masters record and p_transactions to insert or update a transaction.

- p_masters
- p_transactions

Figure A.2 shows the database schema for the Financial Accounting System.



Figure A.2: The database schema for Financial Accounting.

Database Objects Required for Inventory Accounting

The following is a list of objects created that are used by the Inventory Management System developed in <u>Project 3</u> of this book.

Tables

This is a list of tables used by the Inventory Management System. Please note that both the Financial and the Inventory management systems use the tr_header table.

- stock_master
- stock_detail
- tr_header

Triggers

This is a list of the triggers on the stock_detail table.

- update_stk: Update trigger on stock_detail
- insert_stk: Insert trigger on stock_detail
- delete_stk: Delete trigger on stock_detail

Stored Procedures

The Inventory Management System makes use of two stored procedures. The stored procedure p_stock_masters is used to insert or modify an inventory master record and the stored procedure p_stock_trans is used to modify or insert an inventory movement record.



Figure A.3: The database schema for the Inventory Management System.

Appendix B: HailStorm

Passport Authentication is a centralized authentication service provided by Microsoft offering a single sign-in and core profile services for member sites. A typical Web user will have a number of different usernames and passwords scattered over a number of Web sites. It is quite a nuisance to have to remember all these passwords. Users who shop online have to type in their address, phone numbers, credit card information, and what have you, over and over again. It is quite common for shoppers to abandon their shopping cart when they see the long online forms that they have to fill in. Microsoft Passport addresses these problems.

At the time of this writing, Microsoft had just released information and a White Paper on HailStorm. This product is a major technology component of Microsoft's .NET vision and will include Web versions of Hotmail, MSN Messenger, and the Passport user

authentication product. The gist is that Passport will be like an Internet passport. Your critical personal information will be stored at a central location and you will have full control over it. Hence you will use the same password and username over all partner sites. Your personal information will not have to be re-keyed in at these sites. The same would apply to your address book and your bookmarks. These would be accessible from any computer over the Web, and would be created only once.

Passport authentication will be a paid service for sites that make use of the Passport authentication services. At the date of writing, the following companies have decided to use HailStorm and Passport:

- American Express
- Click Commerce
- eBay
- Expedia.com
- Groove Networks

These companies quote receiving the following benefits in using this technology:

American Express

"With Microsoft HailStorm services, American Express can provide their customers greater flexibility and security of purchasing on the Internet and a higher level of customer service. Using their Blue card and special private card reader, American Express can truly validate a customer's identity for purchases on the Internet. The Passport authentication system helps with the validation process. Customers can also reach alerts and notifications via instant messages for bill payment and possible fraudulent activity. They can review the suspected charges and request contact from American Express customer service. Instead of having to wait on hold, the customers receive instant messages to their PC, pager, or cell phone when a customer service representative is available."

Click Commerce

"Through Microsoft HailStorm services, Click Commerce is able to provide manufacturers that use their tools a closer relationship with their partners and customers. The manufacturers can seamlessly update inventory and add new products to the Click Commerce run site. The customers can receive real-time notifications of new products or increased inventory via instant messages to their PC, pager, or cell phone, saving valuable time when searching for products. The manufacturers receive instant notification if their products are back ordered or out of stock on the Click Commerce site. These tools make customer relationships easier to manage for the manufacturers."

eBay

"By utilizing Microsoft HailStorm services, eBay is able to provide their customers realtime tracking of the auctions in which they are participating without requiring them to be logged into eBay's Web site. Through instant messaging, customers can see the latest bids for all of the auctions in which they are participating. They can receive notifications when they have been outbid to their PC, pager, or cell phone, making it easier for them to make a new bid and increase their chances of buying the item. The seller of the item can receive more bids on their item, which increases its selling price. Also, eBay's auction information can be queried through other applications and services to provide customers with the ability to query while not on eBay's site. They can update and add items to auctions through these rich client wizards. With HailStorm services, eBay can keep their customers more directly involved without requiring them to be watching the Web site."

Expedia.com

"Within their prototype, Expedia uses Microsoft's "HailStorm" services such as Passport authentication, Messenger notification, programmatic access to the buddy list, and calendar integration to continue to improve the traveler experience. For example, Expedia utilizes the messenger client for a special tab in which dynamic travel information and offers are displayed. The HailStorm services also help Expedia transform itineraries into communication centers—allowing travelers to pick distinct notification settings for different members of their integrated buddy list. Never again will travelers have to worry about the person picking them up at the airport not knowing their flight status. For trip planning and coordination, Expedia also provides the ability to add itineraries to other buddy list members' calendars."

Groove Networks

"We find HailStorm to be a perfect example of rich, XML Web services that can complement the capabilities of emerging edge-based, peer computing applications such as Groove. In a world where electronic communication is such an important part of our lives, it's critical that people know how to reach us, no matter which communication tools we are using. We believe that HailStorm services will fundamentally improve the user experience for our customers, and as such, we plan later this year to release enhancements to Groove that support the single sign-on and notification infrastructure available to us in Windows XP."

Summary

The instant notification feature is an advantage cited by these companies. You might have used MSN Messenger or Yahoo Messenger. As soon as your friend comes online, you get a beep and a notification from your computer to that effect. HailStorm will use these features of MSN Messenger to flash important notifications to your computer, mobile, and handheld devices. Thus, American Express can flash an instant warning to you when it detects a seemingly fraudulent use of your credit card, and eBay can provide you notifications when you have been outbid in an auction in which you are participating. These messages will be flashed to any HailStorm-connected device and urgent messages do not have to wait until the next time you use a custom device. The next version of Windows—Windows XP—will automatically connect you in Passport when you log in to a computer. Hence, in effect, you will always be ready to receive notifications through HailStorm.

HailStorm will be based on SOAP and XML and will expose a number of functions (services) that developers can consume. Thus, if American Express wants to flash a notification to a user's screen, it will call the appropriate function exposed by HailStorm.

Initially the following functions will be exposed by HailStorm:

- myAddress: electronic and geographic address for an identity
- myProfile: name, nickname, special dates, picture
- myContacts: electronic relationships/address book
- myLocation: electronic and geographical location and rendezvous
- myNotifications: notification subscription, management, and routing
- myInbox: inbox items like e-mail and voice mail, including existing mail systems
- myCalendar: time and task management
- myDocuments: raw document storage
- myApplicationSettings: application settings
- myFavoriteWebSites: favorite URLs and other Web identifiers
- myWallet: receipts, payment instruments, coupons, and other transaction records
- myDevices: device settings, capabilities
- myServices: services provided for an identity

myUsage: usage report for above services

Web software developers will be offered a kit with which to build XML-based services while Web operators like American Express and eBay will be able to license .NET services for their sites. Passport will be free for users; however, other HailStorm services like notifications will come at a price.

Whereas on one hand companies like American Express have welcomed and adopted HailStorm, Microsoft critics feel that Microsoft Passport is an attempt to make the world's largest and richest consumer database, which Microsoft can "harvest" to its benefit. Intrusion of personal privacy is another fear; information can be shared between member sites, to the detriment of a user. For example, a person who buys books on bankruptcy from Barnes & Noble would not like American Express to have that information when applying for a credit card.

List of Figures <u>Chapter 2: Introducing ASP.NET Web Forms and Controls</u>

Figure 2.1: Page Events. Figure 2.2: HTML Controls. Figure 2.3: Web Controls. Figure 2.4: Panel. Figure 2.5: AdRotator. Figure 2.6: Calendar.

Chapter 3: Using ADO.NET in the .NET Framework

Figure 3.1: Interacting with Data.

Figure 3.2: Action Queries.

Figure 3.3: Filtering a DataView.

Figure 3.4: Reading rows and columns collection of a DataTable.

Figure 3.5: DataReader.

Chapter 4: Data Binding

Figure 4.1: Binding "selection" controls.

Figure 4.2: The DataRepeater.

Figure 4.3: Master1 DataGrid.

Figure 4.4: Masters2 DataGrid.

Figure 4.5: Masters2 DataGrid in Edit Mode.

Figure 4.6: The DataList. Figure 4.7: DataList in Edit Mode. Figure 4.8: Binding to XML Data.

Figure 4.9: Master Child Relationship.

Chapter 5: Input Validation

Figure 5.1: Validation controls.

Chapter 6: User Controls

Figure 6.1: Simple user control. Figure 6.2: Exposing properties in user controls. Figure 6.3: Navigation Menu.

Chapter 7: Custom Controls

Figure 7.1: The Lister.

Figure 7.2: The GenEditAdd control in the Edit mode.

Figure 7.3: The GenEditAdd control in the Add mode.

Figure 7.4: The GenEditAdd component in the Edit mode.

Figure 7.5: The Add Mode.

<u>Chapter 8:</u> Business Objects

Figure 8.1: Simple Business Object. Figure 8.2: Testing the database class.

Chapter 9: Working with ASP.NET Web Services

Figure 9.1: Testing the service. Figure 9.2: A call to the web service using the HTTP Get protocol. Figure 9.3: A call to the web service using the HTTP Post protocol. Figure 9.4: Compiling the proxy. Figure 9.5: Testing using the SOAP protocol. Figure 9.6: New project. Figure 9.7: New Web Service. Figure 9.8: Web Access Failed dialog box. Figure 9.9: The Solution pane displaying the four default files. Figure 9.10: Creating the web service. Figure 9.11: Building the solution. Figure 9.12: Testing the functions. Figure 9.13: Setting a breakpoint. Figure 9.14: Using the debugger. Figure 9.15: New Project pane. Figure 9.16: Design of the web form. Figure 9.17: Adding a Web reference. Figure 9.18: Web service discovery. Figure 9.19: Reference to web service in Solution Explorer. Figure 9.20: Testing the Add function. Figure 9.21: Calling the Populate function using Behavior.

Figure 9.22: Enabling the Access data sources across domains option of IE.

Chapter 10: ASP.NET Applications

Figure 10.1: New virtual directory in IIS.

Figure 10.2: Create button.

Figure 10.3: Remove button.

Figure 10.4: New virtual directory in Personal Web Server.

Figure 10.5: Initial request.

Figure 10.6: Page refresh.

Figure 10.7: Session abandoned.

Figure 10.8: Application state.

Figure 10.9: Starting ASPState service.

Figure 10.10: State maintained in SQL Server.

<u>Chapter 12:</u> Tracing

Figure 12.1: Page-level tracing.

Figure 12.2: Using Trace.axd to see the trace output.

<u>Chapter 13:</u> Security

Figure 13.1: The login page.

Figure 13.2: A successful login.

Figure 13.3: Enabling Basic security.

Figure 13.4: Setting permissions in Windows NT.

Figure 13.5: Windows-based login.

<u>Chapter 14:</u> The Design of the Personal Finance Manager

Figure 14.1: The database schema for the Personal Finance Manager application.

<u>Chapter 15:</u> Chart of Accounts

Figure 15.1: The Masters web form.

Figure 15.2: The Masters web form in the add mode allows creation of new records. Figure 15.3: The Masters web form in the edit mode allows you to modify existing records.

Chapter 16: Transactions

Figure 16.1: The Selection form allows you to select a bank or cash account. Figure 16.2: The Transactions form.

Figure 16.3: The Transactions form in Add mode.

Chapter 17: The Trial Balance Report

Figure 17.1: The Trial Balance.

<u>Chapter 18:</u> Creating a Generic Database Web Service

Figure 18.1: Interacting with the web service.

Chapter 19: Designing a Navigation System

Figure 19.1: Navigation links.

Chapter 23: The Design of the Inventory Management Svstem

Figure 23.1: The database schema for the Inventory Management System.

Chapter 24: Inventory Masters

Figure 24.1: The Inventory Masters web form.

Figure 24.2: The Inventory Masters web form in the Add mode.

Figure 24.3: The Inventory Masters web form in the Edit mode.

Chapter 25: Inventory Movements

Figure 25.1: The Inventory Transactions form.

Figure 25.2: The Inventory Transactions form in Add mode.

Figure 25.3: The Inventory Transactions form in Edit mode.

<u>Chapter 26:</u> The Inventory Balances Report

Figure 26.1: The inventory balances report.

<u>Chapter 27:</u> Using the GenEditAdd Control

Figure 27.1: Masters.aspx with edit, add, and delete links. Figure 27.2: GenEditAdd in the Edit mode allows modification of records.

Figure 27.3: The Add mode of GenEditAdd allows insertion of new records.

<u>Chapter 29:</u> Displaying Database Data Using a Strongly-

Typed DataSet

Figure 29.1: Creating a new ASP.NET Web Application.

Figure 29.2: VS will create a Virtual Directory in IIS.

Figure 29.3: The blank project created by VS.

Figure 29.4: The Project Properties.

Figure 29.5: The Data section of the toolbox.

Figure 29.6: The DataLink properties pane.

Figure 29.7: The Choose a query window.

Figure 29.8: Typing in a SQL query in the Generate the SQL Statement window.

Figure 29.9: The Query builder allows you to graphically build queries.

Figure 29.10: The SqlDataAdapter and SqlConnection created in VS.

Figure 29.11: The Server Explorer allows you to explore and manipulate database objects.

Figure 29.12: The Properties page for the SqlConnection.

Figure 29.13: TheProperty Page for the SqlDataAdapter.

Figure 29.14: The Generate DataSet dialog box.

Figure 29.15: An instance of the DataSet is created. by VS.

Figure 29.16: The final output.

<u>Chapter 30:</u> Writing CRUD Applications with Visual Studio.NET

Figure 30.1: Setting the BackGround color of the web form.

Figure 30.2: The stock detail table in the Server Explorer.

Figure 30.3. The SqlConnection and SqlDataAdapter generated by Visual Studio.

Figure 30.4: Generating the DataSet.

Figure <u>30.5</u>: Setting the Properties of the DataGrid.

Figure 30.6: Selecting the columns of the DataGrid.

Figure 30.7: Selecting the Edit, Update, Cancel buttons.

Figure 30.8: Setting the AutoGenerateColumn attribute to False.

Figure 30.9: The Stock Master form displaying data.

<u>Chapter 31:</u> Creating a Web Service Using Visual Studio.NET

Figure 31.1: Creating a new C# ASP.NET web service.

Figure 31.2: The test page for the web service.

Figure 31.3: The Populate method returns database rows as XML.

Figure 31.4: Creating a new C# ASP.NET Web Application.

Figure 31.5: The two projects displayed in the Solution Explorer.

Figure 31.6: Design of the client web form.

Figure 31.7: Web service discovery.

Figure 31.8: Reference to web service in the Solutions Explorer.

Figure 31.9: Selection of a SELECT query displays the results in a DataGrid.

Figure 31.10: The Test RunSql button inserts a random number in the table.

<u>Appendix A:</u> Installing the Sample Database

Figure A.1: create.sql is shown loaded in the isql utility. Figure A.2: The database schema for Financial Accounting.

Figure A.3: The database schema for the Inventory Management System.

List of Tables

Chapter 14: The Design of the Personal Finance Manager

Table 14.1: Relationship between Types and Groups

Table 14.2: The Groups Table Definition

Table 14.3: The Predefined Groups

Table 14.4: The Masters Table

Table 14.5: The tr_header Table

Table 14.6: The Transactions Table

Table 14.7: The tblSelection Table

<u>Chapter 16:</u> Transactions

Table 16.1: Rules for Deposits and Withdrawals Table 16.2: Closing Balance Calculations

Table 16.2: Closing Balance Calculations

<u>Chapter 23:</u> The Design of the Inventory Management

System

<u>Table 23.1:</u> The *stock_master* Table <u>Table 23.2:</u> The *tr_header* Table

Table 23.3: The stock_detail Table

<u>Chapter 25:</u> Inventory Movements

Table 25.1: Closing Balance Calculations

<u>Chapter 27:</u> Using the GenEditAdd Control

Table 27.1: GenEditAdd Quick Reference

<u>Chapter 28:</u> Extending the GenEditAdd Control

Table 28.1: The Four Drop-Down List Sub-properties

List of Examples

ActionQueries.aspx

p authors

Chapter 2: Introducing ASP.NET Web Forms and Controls

<u>State.asp</u> <u>State.aspx</u> <u>Events.aspx</u> <u>events_cb.aspx</u> <u>Events_cb.vb</u> <u>htmlControls.aspx</u> <u>web_intrinsic.aspx</u> <u>panel.aspx</u> <u>adrotator.aspx</u> <u>Calendar.aspx</u> <u>Calendar.aspx</u> <u>MastersGrid.aspx</u> <u>OpenExplicit.aspx</u>

Parameters.aspx Execute.aspx DataView.aspx Collection.aspx DataReader.aspx DataRelation.aspx Chapter 4: Data Binding DataBind.aspx Repeater.aspx Masters1.aspx Masters1.vb Masters2.aspx Masters2.vb PagingSorting.aspx PagingSorting.vb GroupsDlist.aspx GroupsDlist.vb navXML.aspx MasterChild.aspx MasterChild. vb Chapter 5: Input Validation validate.aspx Chapter 6: User Controls simpleUC1.aspx simpleUC2.aspx simpleUC2.ascx nav.XML Navigation.aspx nav.ascx Chapter 7: Custom Controls hello.vb makeVb.bat hellovb.aspx helloC.cs makec.bat HelloC.aspx The Config Masters.aspx form Step1.aspx Step2.aspx GenTestStep3.aspx Step3.vb GenEditAdd.vb Masters.aspx Chapter 8: Business Objects BasicObj.aspx BasicObjC.cs BasicObiC.aspx SQLClass.vb TestVbClass.aspx SQLClassC.cs SQLClassC.bat TestcClass.aspx Chapter 9: Working with ASP.NET Web Services BasicService.asmx

WSDL extract for HTTP Get basicHTTPGet.html WSDL extract for HTTP Post basicHTTPPost.html

mbasicService.bat basicSoap.aspx Behavior.asmx Populate.html AddFunction.html Chapter 10: ASP.NET Applications Global.asax GlobalTest.aspx Global.asax ApplicationState.aspx Session.aspx AppSettingVb.aspx AppSettingC.aspx web.config for Custom Errors HandleError.aspx ErrorTest.aspx handler.vb make.bat Chapter 11: Caching MastersGrid.aspx querystring.aspx <u>nav.xml</u> nav.ascx Navigation.aspx Chapter 12: Tracing Visual Basic.NET <u>C#</u> Trace page.aspx Chapter 13: Security web.config encrypt.aspx Login.aspx default.aspx login.aspx (database version) Chapter 15: Chart of Accounts Stored Procedure p masters Grid1 Update Insert Logic Is the Form The add click Sub Sub Grid1 delete Sub RunSal Masters3.aspx Masters3.vb (Code Behind) Chapter 16: Transactions Stored Procedure p trans insert mstr update mstr delete mstr Selection.aspx Transactions.aspx Page Load Event Sub UpdateSelection Sub ReBind Sub add show Sub add click Sub Grid1 Update The Delete Sub transactions.aspx Transactions.vb (Code Behind)

<u>Chapter 17:</u> The Trial Balance Report

TrialBalance.aspx

Chapter 18: Creating a Generic Database Web Service

SQLService.asmx msqlProxy.bat SQLService.aspx

<u>Chapter 19:</u> Designing a Navigation System Nav.xml

nav.ascx

Chapter 20: Incorporating Web Services in the Chart of

Accounts Form

<u>The Original ReBind Method</u> <u>The Mmodified ReBind Method that Uses the Web Services</u> <u>The Original RunSql Method</u> <u>Modified Version of the RunSql Method</u> Masters3.vb

<u>Chapter 21:</u> Incorporating Web Services in the Transactions

Form

<u>The Original ReBind Method</u> <u>The Modified ReBind Method</u> <u>The Original RunSql Function</u> <u>Modified Version of the RunSql Function</u> Transactions.vb

Chapter 22: Incorporating Web Services in the Trial Balance

<u>The Original ReBind Function</u> <u>The Modified ReBind() Function</u> The Modified TrialBalance.aspx

<u>Chapter 23:</u> The Design of the Inventory Management

System

Nav.xml

Chapter 24: Inventory Masters

Stored Procedure p_stock_masters The ReBind Function Grid1_update Insert Logic in the Form The add_click Sub Sub Grid1_delete Sub RunSql StockMaster.aspx StockMasters.vb

<u>Chapter 25:</u> Inventory Movements

Stored Procedure p_stock_trans Insert_stk Update_stk delete_stk Sub_ReBind Sub_add_show Sub_add_click Sub_Grid1_Update Sub_RunSQL Grid1_delete StockTrans.aspx StockTrans.vb

Chapter 26: The Inventory Balances Report

The ReBind Function StockBalances.aspx

<u>Chapter 27:</u> Using the GenEditAdd Control

<u>config master.aspx</u> <u>The GenEditAdd hyperlinks in masters.aspx</u> <u>Extract from Masters.aspx hooking a DataGrid to GenEditAdd</u> <u>Chapter 28:</u> Extending the GenEditAdd Control <u>DropDown explain.aspx</u> <u>Drop-down List Columns in the Update Mode of GenEditadd</u> GenEditAdd.vb

<u>Chapter 30:</u> Writing CRUD Applications with Visual Studio.NET

The Bind Method Grid1_Edit Grid1_Cancel Grid1_Delete Grid1_Update RunSql Add_Click SaveNew_Click

Chapter 31: Creating a Web Service Using Visual Studio.NET

dbService.asmx WebForm1.aspx.cs