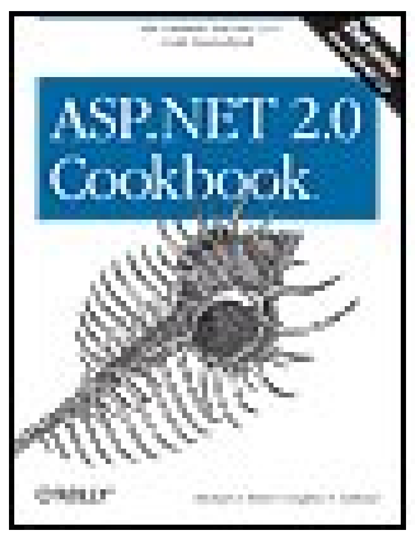


ASP.NET Cookbook, 2nd Edition

By Michael A. Kittel, Geoffrey T. LeBlond

....

Publisher: O'Reilly

Pub Date: December 2005

Print ISBN-10: 0-596-10064-7

Print ISBN-13: 978-0-59-610064-3

Pages: 1014

Overview

Completely revised for ASP.NET 2.0, this new edition of the best-selling *ASP.NET Cookbook* has everything you need to go from beginning to advanced Windows-based web site development using Microsoft's popular Visual Studio 2005 and ASP.NET 2.0 developer tools. Written for the impatient professional, *ASP.NET 2.0 Cookbook* contains more than 125 recipes for solving common and not-so-common problems you are likely to encounter when building ASP.NET-based web applications.

The recipes in this book, which run the gamut from simple coding techniques to more comprehensive development strategies, are presented in the popular Problem-Solution-Discussion format of the O'Reilly Cookbook series. As with the first edition, every solution is coded in both C# and Visual Basic 2005.

Among the additions and revisions to this new edition are:

- Three new chapters, including 25 new recipes for Master and Content pages, Personalization using Profiles and Themes, Custom Web Parts, and more
- New code for every solution, rewritten to take advantage of features and techniques new to ASP.NET 2.0 and available for download

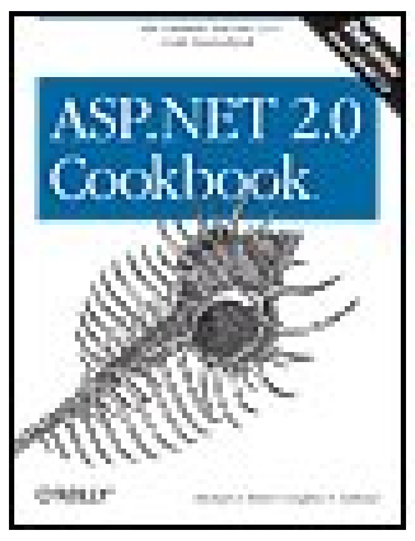
The *ASP.NET 2.0 Cookbook* continues to provide the most comprehensive coverage you'll find anywhere of:

- Tabular controls, including the new GridView control
- Data validation, including the new ASP.NET 2.0 validation controls, as well as techniques for performing your own validation programmatically

- User and custom controls
- Error handling, performance tuning, and caching

Whether you're new to ASP.NET or an experienced Microsoft developer, with*ASP.NET 2.0 Cookbook*, deliverance from a long day (or night) at your computer could be just one recipe away.

PREV



ASP.NET Cookbook, 2nd Edition

By Michael A. Kittel, Geoffrey T. LeBlond

....
Publisher: O'Reilly
Pub Date: December 2005
Print ISBN-10: 0-596-10064-7
Print ISBN-13: 978-0-59-610064-3
Pages: 1014

Table of Contents | Index

PREV

PREV

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Editor:	John Osborn
Developmental Editor:	David Clark
Production Editor:	Matt Hutchinson
Production Services:	GEX, Inc.
Cover Designer:	Emma Colby
Interior Designer:	David Futato
Printing History:	
August 2004:	First Edition.
December 2005:	Second Edition.

ISBN: 0-596-10064-7

[M]

Preface

What This Book Is About

This book is a collection of ASP.NET 2.0 recipes that aims to help you quickly and efficiently solve many of the day-to-day problems you face when developing web applications with the .NET platform. Our recipes run the gamut from simple coding techniques to more comprehensive development strategies that the most experienced ASP.NET programmers will savor. Revised and updated for ASP.NET 2.0, the *ASP.NET 2.0 Cookbook* is your ultimate ASP.NET 2.0 code sourcebook.

> This book is tailored to ASP.NET 2.0. Thus, the vast majority of the code will not run in the ASP.NET 1.x environment. If you're working with ASP.NET 1.x, consider instead O'Reilly's *ASP.NET Cookbook* (the first edition of this book).

More than a compilation of tips and tricks, the *ASP.NET 2.0 Cookbook* solves realworld programming problems and is rooted in our experience as professional programmers who have designed and built richly functional web-based projects for various corporate clients. We think we know the kinds of problems that you face, and we aim to help you solve them.

The *ASP.NET 2.0 Cookbook* contains dozens of code examples, ranging from relatively simple 10-liners to comprehensive, multipage solutions. Without solid and complete working examples, it's difficult to make an informed decision about whether an approach is the right one or whether you should be looking elsewhere. We are convinced that reading good example code is the best path to understanding any development platform, so we've included lots of it and commented our listings to help you follow the logic.

The *ASP.NET 2.0 Cookbook* is written in classic O'Reilly Cookbook style to focus directly on problems you face today or are likely to face in the future. Using a problem-and-solution format, we make it easy for you to skim for a near match to your particular problem. We have pared down the headings to a bare minimum so you can assess if a recipe is pertinent.

Many of us occasionally browse through cookbooks looking for new recipe ideas or exploring the nuances of a culinary style. Similarly, we hope you find this book sufficiently interesting to browse, because in many respects, it is as much about software techniques and methodology as it is about ASP.NET. For example, we offer a full course of error-handling recipes because we believe the topic i important to our audience and there isn't enough information about it in general circulation. We coul have restricted our discussion to page-level error handling, but that seems inadequate to us. Instead we prefer to help you deal with error handling at the application level, a more difficult subject but ultimately more useful to serious developers. We have done the heavy lifting on this and many other important subjects so that you don't have to.

Code reuse is central among the techniques that serious developers consider important, and we've gone to some lengths to illustrate its application in ASP.NET. For example, Chapter 5 includes a recipe that shows how to reuse a code-behind class with another *.aspx* file to provide a different user interface. This is done without any additional coding. As another example, Chapter 4 includes a recipe that sets the focus to a specific control when a page is first loaded, something that could have been accomplished by including some JavaScript in the page's HTML. However, we take a different track by programmatically generating JavaScript client-side code. The reason stems from the type of development work we do, where we are constantly under the gun to quickly generate high-quality code, the ultimate in "short-order cooking." Thus, we are constantly looking for ways to reuse code. By creating a forms library, complete with custom classes that can programmatically generate Java-Script, we are able to build custom forms with a few calls to the library. It's an approach that has proven highly successful for us, and we felt it was important to provide you with a glimpse of it. This is just one of many "reuse-oriented" approaches you'll find in this book.

Who This Book Is For

This book is for journeyman programmers who know the basics of ASP.NET. That said, we're confident that if you are a novice ASP.NET programmer, you will find a home here provided you have done some ASP.NET and either VB or C# programming and are willing to invest time in closely studying our code.

Because this book is not a complete reference for ASP.NET 2.0, it is unlikely to appeal to you if you have merely dabbled in ASP.NET up to this point. You will be better served by first reading a general introduction to ASP.NET programming, such as *Programming ASP.NET 2.0*, by Jesse Liberty and Dan Hurwitz (O'Reilly), where topics are dosed out in measured spoonfuls. After you have mastered the basics, you'll be ready to read this book. We encourage you to look to a general reference or to the MSDN Library when you have routine questions about ASP.NET.

The generous feedback we have received on the first edition of the book tells us that many readers have found the *ASP.NET Cookbook* to be a great way to learn ASP.NET. Some have even recommended that aspiring ASP.NET developers should read it from cover to cover. We are delighted that some readers have used the book this way, but we anticipate that most will reach for this edition only when they need to solve a development problem, so we have organized it for this purpose. Still, we hope you find, as others have, that after months of such use, the book is looking a bit tattered but your ASP.NET code is looking quite good.

How This Book Is Organized

This book is organized into 21 chapters, each of which focuses on a distinct ASP.NET 2.0 topic area.

Chapter 1, *Master Pages*

Master pages are designed to help reduce the replication of identical HTML in your application pages, and the recipes in this chapter show how to make the most of them. The first recipe shows you how to create a quick master/content page arrangement to familiarize you with the approach for using master pages. Next, we cover how to extend a master page's content to include content for other application pages, which is useful when you want your application's login page to appear in one style and the pages that follow it to appear another. The third recipe shows you how to change to another master page without having to edit all of the pages in an application, a common scenario when reuse is important to your application. The last recipe describes how to set the master page programmatically, which can be important when you want to change the application's appearance at runtime. Besides showing you some useful techniques for using master pages, these recipes provide the consistent appearance for all the recipes in the book.

Chapter 2, *Tabular Data*

In its simplest form, displaying tabular data is easy in ASP.NET 2.0: drop in an ASP.NET grid control, connect to the database, and bind it to the control. This is simple enough but it doesn' take long before you realize that the default appearance and behavior of the control is lacking. Indeed, you may even find that the control you've chosen is too bulky, slow, or confusing wher it comes to adding to or modifying its behavior. This chapter helps you make a well-informed decision about which control to use and then provides you with some recipes to solve common development problems as you adapt the control to your liking. Special emphasis is given to ASP.NET 2.0's new `GridView` control, for which we provide several recipes and discussions on the latest ASP.NET 2.0 techniques.

Chapter 3, *Validation*

This chapter provides recipes that perform a number of data validation tasks, such as ensuring that the data a user enters is within a defined range or conform to a specific data type. Other validation recipes show how to ensure that the data entered by the user matches a specific pattern or an entry in a database. You'll learn how to ensure that the user selects an entry in a drop-down list.

Validation groups are new to ASP.NET 2.0, and the last two recipes in the chapter shows you have to take advantage of them. The first shows the basics, and the second goes into more depth by showing you how to handle the validation under programmatic control, which is usefu when you want to perform your own nonstandard validation, such as when you want to check new user's registration against a database.

Chapter 4, *Forms*

The solutions in this chapter provide a series of nonobvious solutions for working with forms. For example, rather than requiring that the user always click a Submit button to send the information on a form to the server, you'll learn how to support the Enter key as well. Another recipe shows how to submit a form to a page different from the current page, which is handy when you want, for example, to have one page that collects form data and a second page that processes the data. Still another shows how to create what appears from the user's perspective as a multipage form but which is actually a single page that uses the Wizard contro new to 2.0a useful technique when you want to keep all of your code in one cohesive unit. Nex you'll learn how to set the focus to a specific control when a page is first loaded, an easy solution for JavaScript but one that is more complicated (and more useful) when you focus on code reuse. Finally, you'll learn how to set the focus to the first control on your form that has a validation error.

Chapter 5, *User Controls*

User controls are a way of encapsulating other controls and code into reusable packages. This chapter shows some ways to use these time- and work-savers to share the same header across multiple pages, display a navigation bar that appears customized on each page by setting properties, and reuse identical code-behind classes within different pages while changing the user presentation in the process. Another recipe shows how to communicate between user controls using event delegates, a handy technique, for example, when you want an action taken with one control to affect multiple other controls. The final recipe in this chapter shows how to programmatically load a user control at runtime, which allows you to customize web page content based on a user's selection.

Chapter 6, *Custom Controls*

The recipes in this chapter center on custom controls, for which you can build your own user interface and add your own backend functionality through the methods, properties, and events that you implement for the controls. For instance, one recipe shows how to combine two or more controls into a single custom control. Another recipe shows how to create a custom control that has HTML-style attributes, which can be handy for customizing the control when it is used in a page. A recipe for creating a custom control maintains state between postbacks, like the server controls in ASP.NET. Another recipe shows how to customize an ASP.NET `TextBox` server control to allow only numeric input.

Chapter 7, *Maintaining State*

The recipes in this chapter are all about maintaining state at the application, session, and page levels, all for the purpose of improving the user's experience. For instance, one recipe shows you how to maintain information needed by all users of an application by placing it in the `Application` object. Another shows how to maintain information about a user throughout a session, the advantage here being that you avoid accessing the database each time the data is needed. Another recipe shows how to preserve small amounts of information on a page between postbacks, which is useful when a page has multiple states and it needs to remember the current state value between postbacks to display properly. Another recipe shows you how to persist complex object information between requests for a page, a useful technique when the page is complex and you don't want to use a database to preserve the information.

Chapter 8, *Error Handling*

This chapter covers error handling at different levels of detail. For example, one recipe shows how to provide robust error handling in methods by taking the best advantage of .NET structured exception handling for dealing with errors. A page-level error-handling recipe shows you how to trap any error that occurs on a page and then, using a page-level event handler, redirect the user to another page that displays information about the problem. Another recipe shows how to handle errors at the application level to log the error information and perform a redirect to a common error page. The final recipe shows how to log detailed messages for debugging, while displaying friendly messages to the user.

Chapter 9, *Security*

Security can be handled in different ways and at different levels in ASP.NET, and this chapter provides recipes that delve into some of the most common solutions. For instance, the first twc recipes show you how to use Forms authentication to restrict access to all or only some pages of an application. Another recipe shows you how to restrict access to pages by the user's role. One recipe shows you how to use Windows authentication, which is useful when all users have accounts on your LAN. The final recipe in the chapter shows you how to use ASP.NET 2.0's Membership and Role providers to secure your web site without having to write any code.

Chapter 10, *Profiles and Themes*

ASP.NET 2.0 has a host of new profile features that are this chapter's recipe subjects. The first recipe shows you how to include user profile data in your application without having to write any code to persist it. The second recipe shows you how to inherit a profile, which is useful when you want to use the same profile definition in multiple applications. The next recipe shows you how to store profile information for users who are not logged in to your application and how to create a mechanism to remove user profile data periodically that is no longer being used.

Themes are another important feature that is supported in ASP.NET 2.0. The penultimate recipe in the chapter shows you how to create an ASP.NET 2.0 theme and how to store a selected theme in a profile. The last recipe shows you how to manage user-personalized themes.

Chapter 11, *Web Parts*

Web parts are the new building blocks of personalization in ASP.NET 2.0, and this chapter will take you a long way down the learning curve in using them in your applications. We start with a recipe that shows you how to use ASP.NET server and user controls as web parts, which is the equivalent of "hello world" for web parts. Creating a reusable web parts catalog is the subject of the next recipe. Creating such a catalog provides you with the ability to have a user control that is a list of controls reused on many pages of an application without having to declare the controls in each page. Creating a custom web part is the subject of the third recipe which is useful when you need some functionality for your web parts that you cannot get with user controls or standard ASP.NET server controls, such as building your web part into a separate assembly for sharing with other applications. Another recipe shows you how to communicate between web parts, which is handy, for example, when one web part needs to act as a filter for another web part. Persisting personalized web part property settings is the subject of the last recipe in the chapter, which is beneficial when you have created a web part with custom properties and you want the property data to be persisted along with the other web part personalization data, so that they are available the next time a user revisits a page.

Chapter 12, *Configuration*

The recipes in this chapter deal with how to configure your applications. For instance, one recipe shows you how to change the default HTTP runtime settings in*web.config*, as a way to familiarize yourself with this file, its contents, and its purpose. Another recipe shows you how to add your own custom application settings to *web.config* (by adding an `<appSettings>` element). Still another shows you how to display custom error messages by adding a `<customErrors>` element to *web.config*. Configuring your application to maintain session state across multiple web servers is the focus of another recipe; this is useful because it shows you how to share session state across a load-balanced web farm or web garden. Another recipe shows you how to read configuration data from something other than the`<appSettings>` element of *web.config* (by using the `WebConfigurationManager`), a useful technique when you need it but, surprisingly, one that is not covered anywhere in the ASP.NET documentation. The chapter's final recipe covers how to add your own configuration elements to*web.config*, which is valuable when the predefined configuration elements provided by ASP.NET are not enough.

Chapter 13, *Tracing and Debugging*

The recipes in this chapter will help you ensure that your applications work as anticipated in their first release, through effective use of testing and debugging. For example, one recipe shows you how to identify the source of page-level problems, such as a slowly performing page. Another shows you how to use tracing to identify application-wide problems without having to modify every page or disrupt output. Still another shows you how to dynamically turr on page tracing to pinpoint the cause of an exception error. Identifying problems within web application components is addressed by another recipe. A follow-up recipe shows you how to identify problems within dual use (non-web-specific) components, which requires a slightly different approach to avoid breaking the component when it is used outside ASP.NET applications. The final few recipes deal with writing trace data to the event log with controllable levels, sending trace data via email with controllable levels, and using a breakpoint to peer intc an application when a condition is met.

Chapter 14, *Web Services*

XML web services are a marquee feature of .NET, and the recipes in this chapter will help you create and consume them. You will find a recipe for creating a web service that returns a custom object, a handy approach when none of the .NET data types meets your needs. Another recipe shows you how to control the URL of a web service at runtime.

Chapter 15, *Dynamic Images*

When working with a creative design team, you may run into the situation where a design uses images for buttons but the button labels need to be dynamic. A recipe in this chapter will show you how to deal with this situation with aplomb by drawing button images on the fly. Another recipe shows you how to create bar charts on the fly from dynamic data. Displaying images stored in a database is the focus of another recipe, which shows you how to read an image from the database and then stream it to the browser. The final recipe in the chapter shows you how to display a page of images in thumbnail format.

Chapter 16, *Caching*

ASP.NET provides the ability to cache the output of pages or portions of pages in memory to reduce latency and make your applications more responsive. If pages are completely static, it's a simple decision to cache them. But if the pages change as a function of query string values o are dynamically created from a database, the decision to cache is not so straightforward. The recipes in this chapter will help you sort through these issues. An additional recipe delves into caching pages based on the browser type and version. A follow-up recipe discusses how to cache pages based on your own custom strings, which gives you, for example, the ability to cache a page based on the browser type: the major version (integer portion of the version number) and the minor version (the decimal portion of the version number). You will find a recipe that shows you how to cache pages based on database dependencies, which is importan when the data on a page is retrieved from a database. Two additional recipes show you how to cache user controls and application data. The penultimate recipe in the chapter shows you how to cache application data based on database dependencies, and the last recipe deals with how to cache data sources.

Chapter 17, *Internationalization*

The recipes in this chapter show you the basics of how to internationalize your applications. Fo instance, the first recipe shows you the necessary *web.config* settings to inform the browser of the character set to use when rendering the application's pages. Another recipe shows you how to support multiple languages in an application without having to develop multiple versions of pages. The final recipe shows you how to override currency formatting, which can be handy, for example, when you want all the text displayed in the user's language but currency values displayed in a specific format, such as U.S. dollars.

Chapter 18, *File Operations*

The recipes in this chapter focus on how to download files from and upload files to the web server. One recipe shows you how to process an uploaded file immediately without storing it or the filesystem, which is handy when you want to avoid the potential pitfalls of uploaded files having the same name or filling the hard drive, or when dealing with the security aspects of allowing ASP.NET write privileges on the local filesystem. The final recipe in the chapter shows you how to store the contents of an uploaded file in a database, which is a useful approach when you want to upload a file to the web server but process it later.

Chapter 19, *Performance*

You will find that performance is a running theme throughout the course of this book. Nevertheless, there are a handful of topics in ASP.NET for which performance warrants seperate discussion. For instance, as discussed in one recipe, you can often reduce a page's size (and improve the page's performance) by disabling the `ViewState` for the page or for a set of controls on the page. Another recipe deals with the oft-mentioned topic of using a `StringBuilder` object instead of the classic string concatenation operators (`&` or `+`) to accelerate string concatenation. The difference in this recipe is that we give you some tangible measures to help you understand how important this approach is and when you can do without it. Another recipe illustrates how to get the best performance out of your application when you ar using read-only data access. Another recipe shows you how to use SQL Server Managed Provider to get the best performance when accessing SQL Server data. Again, we give you some tangible measures, and the results are amazing.

Chapter 20, *HTTP Handlers*

An HTTP handler is a class that handles requests for a given resource or resource type. For instance, ASP.NET has built-in HTTP handlers that process requests for *.aspx, .asmx, .ascx, .cs, .vb*, and other file types. This chapter provides recipes for creating your own custom HTTP handler, which is a useful approach anytime you want to handle an HTTP request for a resourc on your own. For example, one recipe retrieves image data from a database and sends the image data to a browser. A second recipe shows how to create a file download handler.

Chapter 21, *Assorted Tips*

This chapter contains a handful of recipes that do not fit conveniently into other chapters of the book. There are recipes for accessing HTTP-specific information in classes, executing external applications, transforming XML to HTML, determining the user's browser type, dynamically creating browser-specific stylesheets, saving and reusing HTML output, and sending mail.

Topics Not Covered

Though the *ASP.NET 2.0 Cookbook*'s coverage is wide-ranging, there are many topics we don't cover For instance, we won't teach you the basics of XML. (Many other books do a fine job of it, including O'Reilly's *XML in a Nutshell*.) Rather, we use XML in many examples throughout the book and assume you know the basics. Likewise, we apply a similar standard when discussing the fundamentals of object-oriented development and other base-level programming topics.

In a similar vein, we avoid interesting topics that are useless in solving day-to-day development problems. For example, we omitted Passport authentication. Though interesting, the use of Passport authentication is not widespread. Indeed, we have cast away more ideas than we can name because they seemed somehow "off target" for our audience.

Many .NET-related topics are interesting from a programmer's point of view but are not pertinent to ASP.NET 2.0. For instance, working with the .NET process model is a career topic, but a programmer's ability to control it through ASP.NET 2.0 is limited. As a practical matter, topics of this kind didn't make the cut.

Finally, because our main focus is ASP.NET 2.0, we do not delve into the details of using Visual Studi 2005. For the most part, we assume you know the basics of Visual Studio and will only mention something about how to use Visual Studio when we think it is obscure or pertinent.

Sample Source Code

The full source code for the recipes in this book can be found at http://www.dominiondigital.com/AspNetCookbook2/. Samples are provided in VB.NET and C# along with working versions of all the recipes that involve code. The web site will give you all the details on how to download the source code.

Sample Database, Scripts, and Connection Strings

The database used for all samples is a SQL Server 2000 database. It can be found at http://www.dominiondigital.com/AspNetCookbook2/.

As you browse the different sample programs throughout the book, you will see our preferred method of data access is OLE DB. From our experience of writing applications for corporate clients, we have learned that many projects start out using SQL Server but migrate to other database servers, such as Oracle or DB2, during the life of the project. By using OLE DB for data access, we gain the necessary layer of abstraction to switch database platforms easily when and if we need to, without having to change the underlying source code. Nevertheless, when you know that SQL Server will be your database platform for the long run, you will want to use SQL Provider to improve its performance, and Recipe 19.4 will show you how.

The connection string used in the samples throughout the book is shown next. It designates the server as `localhost`, the database as `ASPNetCookbook2`, the database user as `ASPNetCookbook_User`, and the password for the user as `work`. You will need to change these parameters to conform to your database installation.

```
Provider=SQLOLEDB;Data Source=localhost;
Initial  Catalog=ASPNetCookbook2;
UID=ASPNetCookbook_User;
PWD=w0rk"
```

Conventions Used in This Book

Throughout this book, we've used the following typographic conventions:

Italic

This character style is used for emphasis as well as for online items and email addresses. It also indicates the following elements: commands, file extensions, filenames, directory or folder names, and UNC pathnames.

`Constant width`

This character style in body text indicates language constructs, such as the names of keywords, constants, variables, attributes, objects, methods, events, controls, and HTML or XML tags.

`Constant width bold`

This character style is used to call your attention to lines of source code that are especially pertinent in a recipe.

`Constant width italic`

This character style indicates replaceable variables in examples. When a variable appears in this style it is a signal to you that the variable needs to be replaced by contents of your own choosing.

Indicates a VB code snippet.

Indicates a C# code snippet.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book doesn't require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation *does* require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*ASP.NET 2.0 Cookbook*, Second Edition, by Michael A. Kittel and Geoffrey T. LeBlond. Copyright 2006 O'Reilly Media, Inc., 0-596-10064-7."

If you feel your use of code examples falls outside fair use or the permission given here, feel free to contact us at permissions@oreilly.com.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/aspnetckbk22

To comment or ask technical questions about this book, send email to:

> bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

> http://www.oreilly.com

Safari Enabled

Acknowledgments

The authors would like to acknowledge our editors at O'Reilly. First, we remain indebted to Ron Petrusha for giving us the opportunity to write the first edition of this book. We wish to thank John Osborn in recognition of the fact that this edition bears the mark of his extraordinary work on the first edition. For this edition, we would especially like to thank David Clark for making our job so much easier and for his precision and clarity.

We would like to thank our technical editors: Thomas Lewis and Jason Alexander.

We would like to acknowledge the management team at Dominion Digital, Inc., for their ongoing support and their willingness to host the web site for the book.

Finally, we would like to dedicate this book to our children. Michael to Lisa and Julie. Geoff to Claire and Beau.

PREV

Chapter 1. Master Pages

PREV

1.0 Introduction

Most applications use a common HTML design for the majority of their pages, and in the past, developers have conceived many techniques to reduce the replication of the identical HTML in the application pages. However, none of those techniques met the primary goals of providing the HTML i one place or made it easy for non-technical personnel to edit the HTML.

ASP.NET 2.0 provides a new approach with *master pages* that comes close to meeting these primary goals. Master pages contain all of the common HTML, server controls, and code that would normally be replicated in some fashion in the application pages. You place one or morecontent placeholders in the master to reserve the space for page-specific content. *Content pages* provide the content specific to the page and need only reference the desired master page. When a user requests a content page, ASP.NET merges the output of the master page with the output of the content page, resulting in a page that combines the master page layout with the output of the content page.

This chapter provides four recipes for master pages. These include a quick master/content recipe to familiarize you with the technique. Next, we show how to extend a master page's content to include content for other application pages, which you might want to do, for example, when you want your login page to have one appearance and the pages that follow it to build on that appearance. The thir recipe describes how to set the master page for pages within a folder structure without having to set the master page for each content page explicitly. This is handy when you want to use the same master page for the majority of your content pages but change to another master page as needed without having to edit all of the pages in the application. The last recipe describes how to set the master page programmatically, which provides the ability to change how the application appears at runtime; this approach is useful when you want to change the appearance of the rendered pages based on the season or client branding of some sort.

In addition to showing you some useful techniques for using master pages, the recipes found in this chapter serve to provide the consistent appearance for all the recipes in the book. In particular, you will see that we use Recipe 1.1's *ASPNetCookbookVB.master* master page in all the recipes in the book, saving us from having to repeat considerable HTML formatting in all the recipes, something th we were unable to do in the previous edition of this book, which focused on ASP.NET 1.x.

PREV

Recipe 1.2. Generating a Quick Master/Content Page Arrangement

Problem

You want to generate a master/content page arrangement quickly to explore the approach used for mas
pages.

Solution

Create a *.master* file that contains the common HTML for your pages and then create an *.aspx* file that contains the page-specific content.

Create a new master page by following these steps:

1. Select your web site in the Solution Explorer.
2. Right-click and select Add New Item from the context menu.
3. Select Master Page from the Add New Item dialog.

In the *.master* file of the master page:

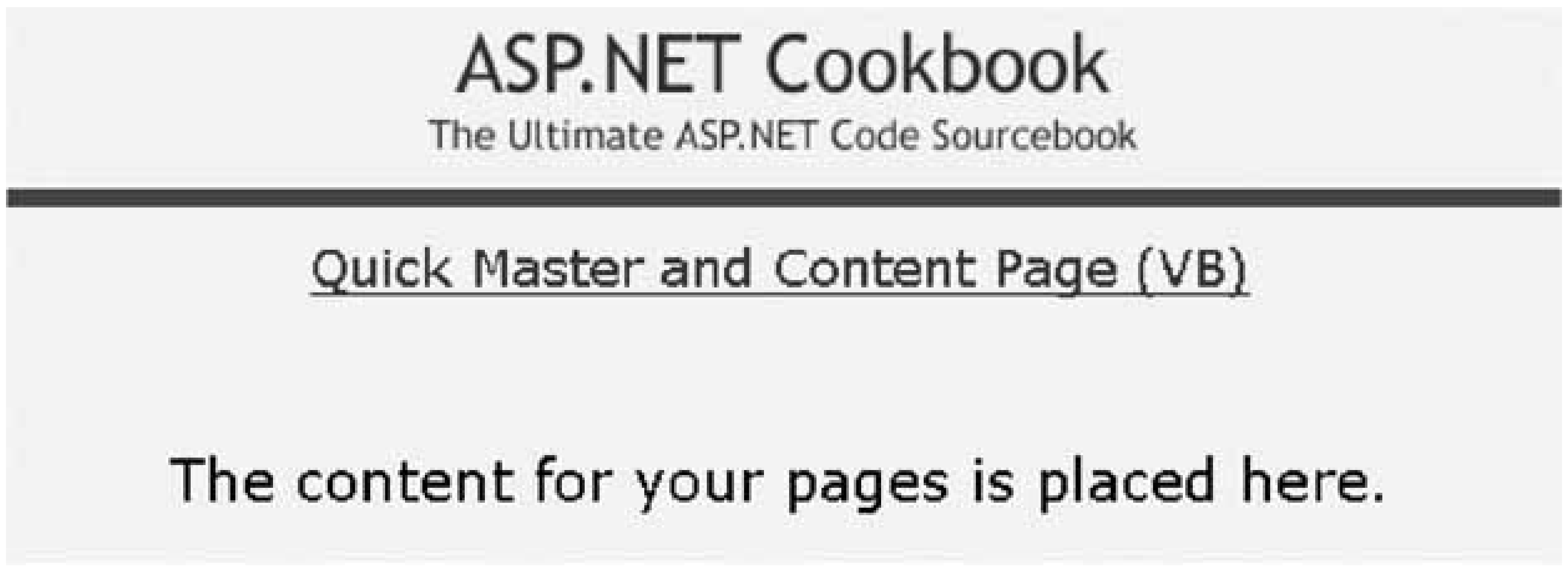
1. Add the HTML that is common for your application pages.
2. Add an `asp:ContentPlaceHolder` control for each portion of the page that will have page-specific content.
3. Set the `ID` attribute to a unique identifier.

In the *.aspx* file:

1. Set the `MasterPageFile` attribute of the `@Page` directive to the name of the *.master* file.
2. Add an `asp:Content` control for each `asp:ContentPlaceHolder` control in the *.master* file.
3. Set the `ContentPlaceHolderID` attribute of the `asp:Content` to the ID of the corresponding `asp:ContentPlaceHolder` in the *.master* file.
4. Add the page-specific content to the `asp:Content` controls.

Figure 1-1 shows the output of the master/content page in our example. Example 1-1shows the *.master*
and Example 1-2 shows the *.aspx* file of our example.

Figure 1-1. Quick master page example output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Quick Master and Content Page (VB)

The content for your pages is placed here.

Discussion

Implementing a basic master/content page arrangement requires no coding. The*.master* file consists of
<%@ Master...%> directive at the top of the page instead of an <%@ Page...%> directive and the comı
HTML (and server controls if required). In addition, it contains one or more`asp:ContentPlaceHolder` cont
that reserve the space for the page-specific content. At a minimum, the`ID` and `Runat` attributes must be
with the `ID` attribute being set to a unique identifier.

```
<asp:ContentPlaceHolder ID="PageBody" Runat="server" />
```

Optionally, the `asp:ContentPlaceHolder` control can contain default content that is displayed if a content
page does not provide any content for the placeholder.

```
<asp:ContentPlaceHolder ID="PageBody" Runat="server" >
  <div align="center">
             <br />
    <br />
    <h4>Default Content Displayed When No Content Is Provided In
                      Content Pages</h4>
        </div>
   </asp:ContentPlaceHolder>
```

At a minimum, the *.aspx* file for the content page contains the`@ Page` directive with the `MasterPageFile`
attribute set to the name of the master page that will provide the template for the content page and an
`asp:Content` control that contains the page-specific content.

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
    AutoEventWireup="false"
     Title="CH01 Quick Master/Content Example" %>
<asp:Content ID="Content1" ContentPlaceHolderID="PageBody" Runat="Server">
          …
</asp:Content>
```

The `ContentPlaceHolderID` attribute of the `asp:Content` control must be set to the `ID` of an `asp:ContentPlaceHolder` control in the master page. Setting the `ID` in this manner identifies which placeholder the content is placed in. Connecting the `asp:Content` control to the `asp:ContentPlaceHolder` control in this manner provides the ability to have multiple placeholders in a master page.

> The *.aspx* files for pages that are linked to master pages do not contain the HTML, head, or body elements. In addition, they are not allowed to have any content outside of the `asp:Content` controls in the page. Placing *any* content outside of the `asp:Content` control results in a parse error when the page is compiled.

In a master/content page arrangement, the page title element is located in the *.master* file. To provide tl ability to set the title for content pages, the `@ Page` directive of the *.aspx* file has a `Title` attribute. Settin the value of the `Title` attribute in the `@ Page` directive results in the title of the rendered page being set the title specified.

See Also

Recipe 1.2

Example 1-1. Quick master/content page (.master)

```
<%@ Master Language="VB"
     AutoEventWireup="false" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
               "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
      <title>ASP.NET Cookbook 2nd Edition</title>
  <link rel="Stylesheet" href="css/ASPNetCookbook.css" alt="ASPNETCookbook"/>
</head>
<body>
      <form id="form1" runat="server">
           <div align="center" class="header">
                <img src="images/ASPNETCookbookHeading_blue.gif" />
           </div>
    <div>
                <asp:ContentPlaceHolder ID="PageBody" Runat="server" >
<div align="center">
```

```
                          <br />
     <br />
     <h4>Default Content Displayed When No Content Is Provided
                                In Content Pages</h4>
                     </div>
                </asp:ContentPlaceHolder>
          </div>
     </form>
</body>
</html>
```

Example 1-2. Quick master/content page (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
       AutoEventWireup="false"
     Title="CH01 Quick Master/Content Example" %>
<asp:Content ID="Content1" ContentPlaceHolderID="PageBody" Runat="Server">
     <div align="center" class="pageHeading">
          Quick Master and Content Page (VB)
     </div>
  <br />
  <br />
  <p align="center">The content for your pages is placed here.</p>
</asp:Content>
```

Recipe 1.3. Extending a Master Page's Content to Include Content for Other Application Pages

Problem

You have a master page that provides the basic content for some of your pages but you would like tc extend the master page to include additional content for other pages. You might want to do this, for example, when you want the login page of your application to appear one way and the pages that follow it to appear in another way that builds on the appearance of the login page.

Solution

Nest master pages as outlined here. Create the base master page that contains the minimum conten used by content pages and then create the content pages that require the content of the base maste page. Next, create another master page that references the base master page and adds additional content required for other pages. Now, create content pages that require the content of master pages and reference the second master page.

In the *.master* file of the base master page:

1. Add the HTML that is common for your application pages.
2. Add an `asp:ContentPlaceHolder` control for each portion of the page that will have page-specific content.
3. Set the `ID` attributes to unique identifiers.

In the *.aspx* file of the pages that will use the base master page:

1. Set the `MasterPageFile` attribute of the `@ Page` directive to the name of the *.master* file of the base master page.
2. Add an `asp:Content` control for each `asp:ContentPlaceHolder` control in the *.master* file of the base master page.
3. Set the `ContentPlaceHolderID` attribute of the `asp:Content` to the ID of the corresponding `asp:ContentPlaceHolder` in the master page.
4. Add the page-specific content to the `asp:Content` controls.

In the *.master* file of the second master page:

1. Set the `MasterPageFile` attribute of the `@ Master` directive to the name of the *.master* file of the base master page.
2. Add the additional HTML that is common for the second type of application pages.
3. Add an `asp:ContentPlaceHolder` control for each portion of the page that will have page-specific content.
4. Set the `ID` attribute to a unique identifier.

In the *.aspx* file of the pages that will use the second master page:

1. Set the `MasterPageFile` attribute of the `@ Page` directive to the name of the second *.master* file.
2. Add an `asp:Content` control for each `asp:ContentPlaceHolder` control in the *.master* file.
3. Set the `ContentPlaceHolderID` attribute of the `asp:Content` to the ID of the corresponding `asp:ContentPlaceHolder` in the second master page.
4. Add the page-specific content to the `asp:Content` controls.

Figure 1-2 shows the output of the page that uses the nested master page in our example. The base master page and a sample content page that uses the master page are shown in Examples 1-1and 1-2 in Recipe 1.1. Example 1-3shows the *.master* file, and Example 1-4 shows the *.aspx* file of this example.

Figure 1-2. Nested master page example output

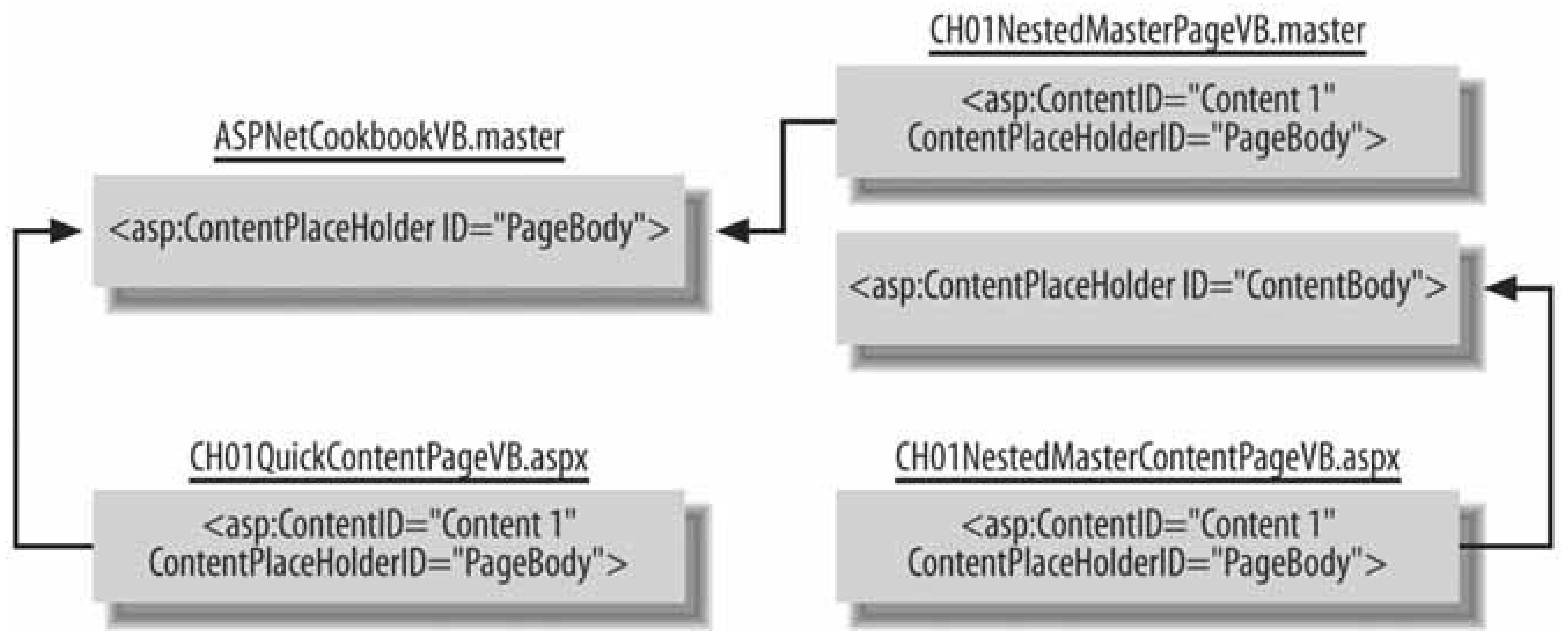
Discussion

In many web applications, the content of the pages is a function of the location within the application A login page, for example, may contain only a header and the required controls for inputting the logi credentials and a button to initiate the login. Pages displayed after login may contain the same header as the login page plus navigational sections under the header and potentially along the left side of the pages. You can use master pages to support this structure and define the HTML only once

In this example, we use the master page defined in Recipe 1.1 as the base master page for nesting. contains a header that will be used for all pages.

In the second master page, we have added a menu to the left side of the page andanother `asp:ContentPlaceHolder` for the page-specific content that will be to the right of the menu. Figure 1-3 shows the hierarchy of the nested master pages in this example.

Figure 1-3. Hierarchy of nested master pages


CH01NestedMasterPageVB.master
<asp:ContentID="Content 1" ContentPlaceHolderID="PageBody">
ASPNetCookbookVB.master
<asp:ContentPlaceHolder ID="PageBody">
<asp:ContentPlaceHolder ID="ContentBody">
CH01QuickContentPageVB.aspx
<asp:ContentID="Content 1" ContentPlaceHolderID="PageBody">
CH01NestedMasterContentPageVB.aspx
<asp:ContentID="Content 1" ContentPlaceHolderID="PageBody">

Content pages that need only a header will reference the base master page (*ASPNet-CookbookVB.master*) and provide the content for the `PageBody asp:ContentPlace-Holder` control. Content pages that need a header and the left menu will reference the second master page (*CH01NestedMasterPageVB.master*) and will provide content for the `ContentBodyasp:ContentPlaceHolder` control.

You can edit simple master pages in the Visual Studio 2005 designer, which provides a WYSIWYG environment. Nested master pages cannot be edited in the designer, however. They must be edited in Source mode. Attempting to access the designer for nested master pages or content pages that use a nested master results in an error with a message indicating the nested master pages cannot be edited in the designer.

Master pages can be as simple or complex as your application dictates. They can include any number of content placeholders and can be nested to any level that meets the needs of your application.

See Also

Recipe 1.1

Example 1-3. Nested master page (.master)

```
<%@ Master Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
       AutoEventWireup="false" %>
<asp:Content  ID="Content1"  ContentPlaceHolderID="PageBody"  Runat="Server">
     <table class="pageBody">
          <tr>
               <td class="leftMenu">
                    <ul class="menuList">
                         <li class="menuListItem">Online Examples</li>
                         <li class="menuListItem">Buy The Book</li>
                         <li class="menuListItem">The Authors</li>
                         <li class="menuListItem">Errata</li>
     <li class="menuListItem">Feedback</li>
                    </ul>
               </td>
               <td class="contentBody">
                    <asp:contentplaceholder id="ContentBody" runat="server" />
               </td>
          </tr>
     </table>
</asp:Content>
```

Example 1-4. Nested master page (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/CH01NestedMasterPageVB.master"
       AutoEventWireup="false"
    title="Nested Master Content Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentBody" Runat="Server" >
     <div align="center" class="pageHeading">
          Nested Master Pages (VB)
     </div>
  <br />
  The content for individual pages is placed here.
</asp:Content>
```

PREV

Recipe 1.4. Changing Which Master Page Is Used Without Modifying All Affected Application Pages

Problem

You want to use the same master page for all pages in a section of your application, but you want the al to change which master page is used without having to modify all of the affected pages, something you cannot do with the more traditional master/content page approaches described in the previous recipes i chapter.

Solution

Implicitly assign a master page to a content page as follows:

1. Create a new folder within your application.
2. Create the master page your application will be using.
3. Place all pages that will be using the master page in the folder.
4. Create a *web.config* file that contains a `<pages>` element with the `masterPageFile` attribute set to th name of the *.master* file and place it in the folder.

Example 1-5 shows the *web.config* file used to set the master page implicitly, and Examples 1-6and 1-7 show the *.master* and *.aspx* files for our example.

Discussion

In some applications, it is desirable to use the same master page for a large number of content pages a the same time, to be able to change to another master page without having to edit all of the pages in th application. ASP.NET 2.0 provides the ability to assign the master page implicitly to content pages by us the new `masterPageFile` attribute of the `<pages>` element in the *web.config* file. Setting the `masterPageFi` attribute to the name of a *.master* file assigns the master page to all content pages in the folder where t *web.config* file is located. This assigns the master page to all content pages in all subfolders unless anot *web.config* file in a subfolder overrides the setting.

ASP.NET provides a lot of flexibility when using this approach. Any content page can still explicitly set th `MasterPageFile` attribute in the `@ Page` directive, as described in Recipe 1.1, which will override the settii the *web.config* file. This is convenient when you have a small number of pages that need to be handled differently; however, this can become confusing when the assignment of a master page is changed in th *web.config* file and the developer is expecting it to affect all pages.

Implicitly assigning a master page using this approach will not work for nested master pages. If a nested master page is included without explicitly setting the `MasterPageFile` attribute of the `@ Master` directive, you will get an error indicating that content controls are only allowed in pages that reference a master page.

In our example, we have created a new folder and placed in it the *web.config* file shown in Example 1-5 a with the *.master* and *.aspx* files shown in Example 1-6 and Example 1-7 . The primary difference betweer this example and the example shown in Recipe 1.1 is the removal of the `MasterPageFile` attribute from th `Page` directive of the content page and the presence of the *web.config* file.

Using this approach can cause problems referencing images and stylesheets. This is particularly an issue if a master page uses images, is located in the root directory, and is then used in a subfolder. The page is rendered using the path to the subfolder resulting in the URL for images pointing to the subfolder. If the images exist only in the root folder, the images referenced by the master page will not be displayed.

See Also

Recipes 1.1 and 1.2

Example 1-5. web.config file for implicitly assigning a master page to a cont page

```
<?xml version="1.0" ?>
<configuration>
     <system.web>
          <pages masterPageFile="~/CH01AttachMaster/ASPNetCookbookVB.master" />
     </system.web>
</configuration>
```

Example 1-6. Implicitly assigning a master page to a content page (.master

```
<%@ Master Language="VB"
    AutoEventWireup="false" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
              "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
      <title>ASP.NET Cookbook 2nd Edition</title>
  <link rel="Stylesheet" href="../css/ASPNetCookbook.css" />
</head>
<body>
      <form id="form1" runat="server">
          <div align="center" class="header">
               <img src="../images/ASPNETCookbookHeading_blue.gif" />
          </div>
          <div>
               <asp:ContentPlaceHolder ID="PageBody" Runat="server" >
<div align="center">
                         <br />
                         <br />
                         <h4>Default Content Displayed When No Content Is Provided In
                                   Content Pages</h4>
                    </div>
               </asp:ContentPlaceHolder>
          </div>
     </form>
</body>
</html>
```

Example 1-7. Implicitly assigning a master page to a content page (.aspx)

```
<%@ Page Language="VB"
    AutoEventWireup="false"
    title="Attaching Master Pages Using Web.Config" %>
<asp:Content ID="Content1" ContentPlaceHolderID="PageBody" Runat="Server">
     <div align="center" class="pageHeading">
          Attaching Master Pages Using Web.Config (VB)
     </div>
  <br />
  <p align="center">The content for your pages is placed here.</p>
</asp:Content>
```

PREV

Recipe 1.5. Setting the Master Page at Runtime

Problem

You have an application where you need to change the assignment of the master pages used at runtime.

Solution

Create the desired master/content pages as described in Recipes 1.1 and 1.2. In the `Page_ PreInit` event handler of code-behind for the content pages, set the `Page.MasterPageFile` property to the name of the desired *.master* file.

Example 1-8 shows the *.aspx* file, and Examples 1-9 and 1-10 show the VB and C# code-behind files for our application.

Discussion

Many applications need the ability to change the appearance of the rendered pages as a function of many factors such as a different look for the seasons or for client branding. Using master pages and changing the assigned master page at runtime can meet the requirements of these applications.

With ASP.NET 2.0, the Page class has a new `MasterPageFile` property that can be set at runtime to change the master page used to render the content page. The `MasterPageFile` property must be set before ASP.NET begins processing the page. The `Page_PreInit` event occurs immediately before the page processing occurs and is the last opportunity to change the master page that will be used to render the page.

Setting the `MasterPageFile` property at any point in the page processing after the `Page_PreInit` event results in an exception being thrown, indicating the `MasterPageFile` property cannot be set after the `Page_PreInit` event.

The `Page` class has a new read-only `Master` property that can be used to access the `MasterPage` object. Through the `MasterPage` object, you can access any public properties or methods of the master page. If you have added public properties or methods to your master pages that you need to access in your content pages, you will need to cast the `MasterPage` object to the class type of your master page as shown below for a `pageHeading` property.

```
CType(Page.Master, ASPNetCookbookVB_master).pageHeading = "Test Heading"
```

```
((ASPNetCookbookCS_master)Page.Master).pageHeading = "Test Heading";
```

In our example, we have stored the name of the master page that will be set at runtime in the *web.config* as shown below. In your application, you might want to obtain the master page information from a database or any other data store that your application needs:

```
<appSettings>
          <add key="runtimeMasterPage" value="CH01MasterPage1VB.master" />
   </appSettings>
```

```
<appSettings>
<add  key="runtimeMasterPage"  value="CH01MasterPage1CS.master"  />
</appSettings>
```

In the `Page_PreInit` event handler, we read the master page name from the *web.config* file and set the `MasterPageFile` property:

```
Private Sub Page_PreInit(ByVal sender As Object, _
                            ByVal e As System.EventArgs) Handles Me.PreInit
     Dim masterPage As String

  'get name of master page from web.config
  masterPage = ConfigurationManager.AppSettings("runtimeMasterPage")

  'set the master page to be used with this page
  Page.MasterPageFile = masterPage
   End Sub 'Page_PreInit
```

```
private void Page_PreInit(Object sender,
                             System.EventArgs e)
   {
     String masterPage = null;

  //get name of master page from web.config
  masterPage = ConfigurationSettings.AppSettings["runtimeMasterPage"];

  //set the master page to be used with this page
  Page.MasterPageFile = masterPage;
   } //Page_PreInit
```

See Also

Recipes 1.1 and 1.2

Example 1-8. Setting the master page at runtime (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
     AutoEventWireup="false"
   CodeFile="CH01SetMasterPageAtRuntimeVB.aspx.vb"
   Inherits="ASPNetCookbook.VBExamples.CH01SetMasterPageAtRuntimeVB"
  title="Set Master Page At Runtime" %>
<asp:Content ContentPlaceHolderID="PageBody" Runat="Server">
     <div align="center" class="pageHeading">
          Set Master Page At Runtime (VB)
     </div>
  <br />
  <br />
  <p align="center">The content for your pages is placed here.</p>
</asp:Content>
```

Example 1-9. Setting the master page at runtime (.vb)

```
Option Explicit On
Option Strict On
Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
  '''  CH01SetMasterPageAtRuntimeVB.aspx
  ''' </summary>
  Partial Class CH01SetMasterPageAtRuntimeVB
    Inherits System.Web.UI.Page

 '''**********************************************************************
 ''' <summary>
 ''' This routine provides the event handle for the page preinit
 ''' event.  It is responsible for setting the master page file
 ''' using a setting is the app settings section of web.config
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Private Sub Page_PreInit(ByVal sender As Object, _
       ByVal e As System.EventArgs) Handles Me.PreInit
      Dim masterPage As String

   'get name of master page from web.config
   masterPage = ConfigurationManager.AppSettings("runtimeMasterPage")

   'set the master page to be used with this page
   Page.MasterPageFile = masterPage
    End Sub 'Page_PreInit
  End Class 'CH01SetMasterPageAtRuntimeVB
End Namespace
```

Example 1-10. Setting the master page at runtime (.cs)

```
using System;
using System.Configuration;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
  ///  CH01SetMasterPageAtRuntimeCS.aspx
  /// </summary>
  public partial class CH01SetMasterPageAtRuntimeCS : System.Web.UI.Page
  {

    ///***********************************************************************
 /// <summary>
 /// This routine provides the event handle for the page preinit
 /// event.  It is responsible for setting the master page file
 /// using a setting is the app settings section of web.config
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 private void Page_PreInit(Object sender,
                                        System.EventArgs e)
    {
      String masterPage = null;

   //get name of master page from web.config
   masterPage = ConfigurationSettings.AppSettings["runtimeMasterPage"];

   //set the master page to be used with this page
   Page.MasterPageFile = masterPage;
    }  //Page_PreInit
   } // CH01SetMasterPageAtRuntimeCS
}
```

PREV

PREV

Chapter 2. Tabular Data

PREV

Introduction

When it comes to displaying tabular data, ASP.NET can save you a lot of time, provided you play you cards right. If you want to display data from a database in a table and you're not concerned about performance or your ability to control the arrangement of the data items within the display, you can whip up something with the `GridView` control in minutes using a few lines of code. Alternatively, if you're more inclined to control the tabular display, you can shape and mold the template-driven `Repeater` and `DataList` controls beyond recognition and still not step far outside a well-trodden path. But like everything else in ASP.NET, you have to know where to start and what the trade-offs are. For instance, it's helpful to know how and when the `GridView` control begins to fall short of the mark. That knowledge will save you the trouble of having to retrace your steps later to implement another tabular control altogether, something we've had to do ourselves at an inopportune moment during a hot project. The recipes in this chapter ought to get you well down the curve with displaying your tabular data and prevent you having to revisit some of our same mistakes.

For readers of the first edition of the *ASP.NET Cookbook*, you should know that we have revised this chapter to incorporate the latest enhancements to ASP.NET 2.0, including, for example, updating all the code to leverage the latest techniques as well as adding several strategies for taking best advantage of the new `GridView` control.

While on the subject of the `GridView` control, you might also have the impression that the new `GridView` control is the all-in-one solution for displaying tabular data. We do not hold with this view; for one thing, the server processing required to build the applicable output is large and may not always be the best approach for your application. What's more, the `GridView` does not afford the flexibility that its predecessors, particularly the `DataGrid`, offer in terms of controlling custom paging. The `GridView` is a useful new control. But, lest you apply `GridView` too broadly to your code, it helps to know its strengths and limitations as we will lay out in the chapter's first recipe.

PREV

Recipe 2.2. Selecting the Right Tabular Control

Problem

You want to use an ASP.NET control to display some data in a tabular format.

Solution

Use a `Repeater, DataList, DataGrid`, or `GridView` control. Always choose the smallest and fastest control that meets your needs, which invariably will be influenced by other criteria as in these examples:

If you need a quick and easy solution

Use a `GridView`.

If you need a lightweight read-only tabular display

Use a `Repeater`.

If you need your solution to be small and fast

Use a `Repeater` (lightest) or `DataList` (lighter).

If you want to use a template to customize the appearance of the display

Choose a `Repeater` or `DataList`.

If you want to select rows or edit the contents of a data table

Choose a `DataList`, a `DataGrid`, or a `GridView`.

If you want built-in support to sort your data by column or paginate its display

Choose a `GridView`.

If you want to use custom pagination

Choose a DataGrid.

Discussion

ASP.NET provides four excellent options for displayingtabular data: Repeater, DataList, DataGrid, and GridView. Each comes with trade-offs. For instance, the GridView control is versatile, but you can pay a price in terms of performance. On the flip side, theRepeater control is lighter weight but is for read-only display; if you later decide you need to edit your data, you will have to rework your code t use the DataList, DataGrid, or GridView control instead (unless, of course, you want to embark on your own custom coding odyssey).

The impact on performance is because ASP.NET creates a control for every element of aDataGrid and GridView control, even white space, which is built as aLiteral control. Each of these controls is responsible for rendering the appropriate HTML output. TheDataGrid and the GridView are, therefore, the heavyweights of the grid control group because of the server processing required to build the applicable output. TheDataList is lighter and the Repeater lighter still.

Table 2-1 summarizes the built-in features supported by the tabular controls and only includes controls that support data binding. (A standard Table control is not included because it does not inherently support data binding though individual controls placed in a table can be data bound.) With custom code, there are almost no limits to what you can do to modify the behavior of these controls.

Table 2-1. Comparative summary of native tabular control features

Feature	Repeater control	DataList control	DataGrid control	GridView control
Default appearance	None (template-driven)	Table	Table	Table
Automatically generates columns from the data source	No	No	Yes	Yes
Header can be customized	Yes	Yes	Yes	Yes
Data row can be customized	Yes	Yes	Yes	Yes
Supports alternating row customization	Yes	Yes	Yes	Yes
Supports customizable row separator	Yes	Yes	No	Yes
Footer can be customized	Yes	Yes	Yes	Yes
Supports pagination	No	No	Yes	Yes
Supports custom paging	No	No	Yes	No
Supports sorting	No	No	Yes	Yes
Supports editing contents	No	Yes	Yes	Yes

Feature	Repeater control	DataList control	DataGrid control	GridView control
Supports selecting a single row	No	Yes	Yes	Yes
Supports selecting multiple rows	No	No	No	No
Supports arranging data items horizontally or vertically (from left-to-right or top-to-bottom)	No	Yes	No	No
Supports sorting and paging using asynchronous callbacks (see Recipe 2.15)	No	No	No	Yes

Performance issues aside, you must consider other aspects when choosing a tabular control. As a rule, the `DataGrid` and `GridView` work well for a quick-and-dirty tabular display (see Recipe 2.2) and for other situations in which you think you'll be reasonably satisfied with its default appearance and behavior. Indeed, because the `DataGrid` and `GridView` are so versatile, this chapter provides many recipes for modifying and adapting them. However, if you anticipate needing a lot of flexibility in controlling the organization and layout of the tabular display or you do not need to edit or paginate the data, you may want to consider using the `DataList` or `Repeater` instead. For example, Recipe 2.3 shows how you can use templates to organize and enhance the output of a tabular display. Take a look at that recipe's output (Figure 2-2) to see what we're driving at. Some upfront planning in this respect can save you considerable time and effort down the road.

Recipe 2.3. Generating a Quick-and-Dirty Tabular Display

Problem

You want to display data from a database in a table, and you're not overly concerned about performance or your ability to control the arrangement of the data items within the display.

Solution

Use a `GridView` control and bind the data to it.

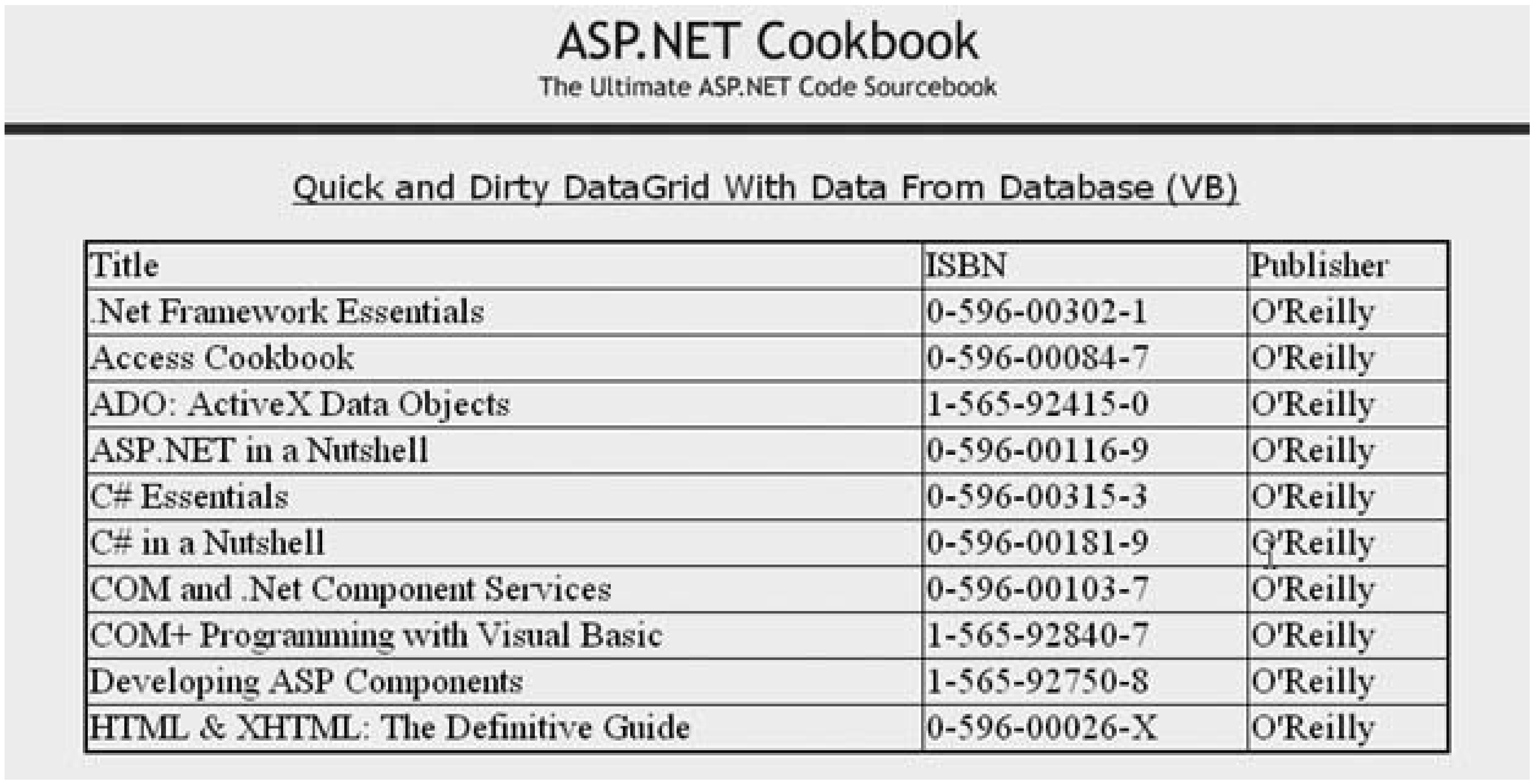
In the *.aspx* file, add the `GridView` control responsible for displaying the data.

In the code-behind class for the page, use the .NET language of your choice to:

1. Create a `SqlDataSource`.
2. Set the `ConnectionString, DataSourceMode, ProviderName`, and `SelectCommand` properties of the `SqlDataSource`.
3. Assign the data source to the `GridView` control and bind it.

Figure 2-1 shows the appearance of a typical `GridView` in a browser. Examples 2-1 through 2-3 show the *.aspx* and VB and C# code-behind files for the application that produces this result.

Figure 2-1. Quick-and-dirty GridView output



Quick and Dirty DataGrid With Data From Database (VB)

Title	ISBN	Publisher
.Net Framework Essentials	0-596-00302-1	O'Reilly
Access Cookbook	0-596-00084-7	O'Reilly
ADO: ActiveX Data Objects	1-565-92415-0	O'Reilly
ASP.NET in a Nutshell	0-596-00116-9	O'Reilly
C# Essentials	0-596-00315-3	O'Reilly
C# in a Nutshell	0-596-00181-9	O'Reilly
COM and .Net Component Services	0-596-00103-7	O'Reilly
COM+ Programming with Visual Basic	1-565-92840-7	O'Reilly
Developing ASP Components	1-565-92750-8	O'Reilly
HTML & XHTML: The Definitive Guide	0-596-00026-X	O'Reilly

Discussion

Implementing a simple `GridView` requires little coding. You must first add a `GridView` control to the *.aspx* file for your application and set a few of its attributes, as shown in Example 2-1. The `GridView` control has many attributes you can use to control the creation of a `GridView` object, but only three are required for this example: the `id, runat`, and `AutoGenerateColumns` attributes. The `id` and `runat` attributes are required by all server controls. When the `AutoGenerateColumns` attribute is set to `true`, it causes the `GridView` to create the required columns automatically along with their headings from the data source.

The code required to read and bind the data to the `GridView` goes into the code-behind class associated with the *.aspx* file as shown in Examples 2-2 (VB) and 2-3 (C#). In our example, this code is placed in the `Page_Load` method for convenience of illustration. It creates a `SqlDataSource`, sets the properties to define the source of the data and the `SELECT` command to retrieve the data from the database, and binds the data source to the `GridView` control. When the `DataBind` method of the `GridView` method is executed, the data source opens a connection to the database, retrieves the data, closes the connection to the database, and then binds the data to the `GridView`.

Setting the `AutoGenerateColumns` attribute of a `DataGrid` to `true` is a simple way to format your data, but it has two drawbacks. First, using the attribute causes a column to be created for every column specified in the `Select` statement, so you should be careful to include only the data you want to see in the `GridView` in the statement. In other words, use the `SELECT *` statement with caution. Second, the columns you `SELECT` will be given the same names as the columns in the database. You can get around this problem by using the `AS` clause in your `SELECT` statement to rename the columns when the data is read into the data reader.

See Also

Recipes 2.14, 2.16, 2.17, and 2.18. For more information on the `GridView` control and `SqlDataSource`, look in the MSDN Library.

Example 2-1. Quick-and-dirty DataGrid (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02QuickAndDirtyGridViewVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02QuickAndDirtyGridViewVB"
 Title=" Quick and Dirty GridView" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">
  Quick and Dirty GridView With Data From Database (VB)
 </div>
 <asp:GridView ID="gvQuick" Runat="Server"
  AutoGenerateColumns="true"
  BorderColor="#000080"
  BorderWidth="2px"
  HorizontalAlign="Center"
  Width="90%" >
 </asp:GridView>
</asp:Content>
```

Example 2-2. Quick-and-dirty DataGrid code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02QuickAndDirtyGridViewVB.aspx
 ''' </summary>
 Partial  Class  CH02QuickAndDirtyGridViewVB
  Inherits System.Web.UI.Page

   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim dSource As SqlDataSource = Nothing

   If (Not Page.IsPostBack) Then
    'configure the data source to get the data from the database
    dSource = New SqlDataSource()
    dSource.ConnectionString = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dSource.DataSourceMode = SqlDataSourceMode.DataReader
    dSource.ProviderName = "System.Data.OleDb"
    dSource.SelectCommand = "SELECT Title, ISBN, Publisher " & _
       "FROM Book " & _
       "ORDER BY Title"

    'set the source of the data for the gridview control and bind it
    gvQuick.DataSource = dSource
    gvQuick.DataBind()
   End If
  End Sub 'Page_Load
 End Class 'CH02 QuickAndDirtyGridViewVB
End Namespace
```

Example 2-3. Quick-and-dirty DataGrid code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
   ///  CH02QuickAndDirtyGridViewCS.aspx
  /// </summary>
   public partial class CH02QuickAndDirtyGridViewCS : System.Web.UI.Page
{
   ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
```

```
    /// <param name="sender">Set to the sender of the event</param>
    /// <param name="e">Set to the event arguments</param>
    protected void Page_Load(object sender, EventArgs e)
    {
    SqlDataSource dSource = null;

    if (!Page.IsPostBack)
    {
     // configure the data source to get the data from the database
     dSource = new SqlDataSource();
     dSource.ConnectionString = ConfigurationManager.
       ConnectionStrings["dbConnectionString"].ConnectionString;
     dSource.DataSourceMode = SqlDataSourceMode.DataReader;
     dSource.ProviderName = "System.Data.OleDb";
     dSource.SelectCommand = "SELECT Title, ISBN, Publisher " +
        "FROM Book " +
        "ORDER BY Title";

     // set the source of the data for the gridview control and bind it
     gvQuick.DataSource = dSource;
     gvQuick.DataBind();
     }
    } // Page_Load
  } //  CH02QuickAndDirtyGridViewCS
}
```

Recipe 2.4. Enhancing the Output of a Tabular Display

Problem

You need to display data from a database in a way that lets you organize and enhance the output beyond the confines of the DataGrid or GridView control's default tabular display. Selecting and editing the data are unimportant as is navigating through the data.

Solution

Use a Repeater control with templates and then bind the data to the control.

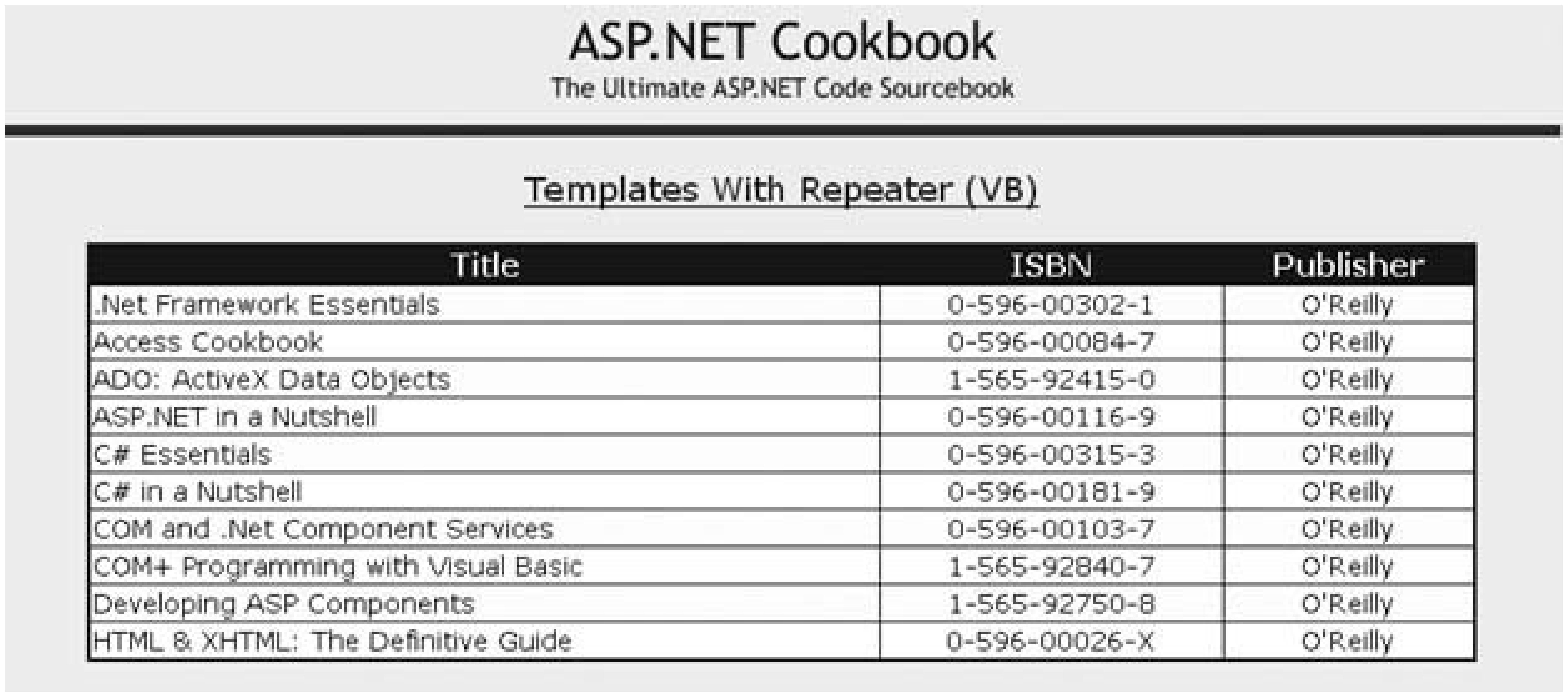
In the *.aspx* file, add a Repeater control and the associated templates for displaying the data.

In the code-behind class for the page, use the .NET language of your choice to:

1. Create a SqlDataSource.
2. Set the ConnectionString, DataSourceMode, ProviderName, and SelectCommand properties of the SqlDataSource.
3. Assign the data source to the Repeater control and bind it.

Figure 2-2 shows the appearance of a typical Repeater in a browser. Examples 2-4, 2-5 through 2-6 show the *.aspx* and code-behind files for an application that produces this result.

Figure 2-2. Using templates with Repeater control display output



ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Templates With Repeater (VB)

Title	ISBN	Publisher
.Net Framework Essentials	0-596-00302-1	O'Reilly
Access Cookbook	0-596-00084-7	O'Reilly
ADO: ActiveX Data Objects	1-565-92415-0	O'Reilly
ASP.NET in a Nutshell	0-596-00116-9	O'Reilly
C# Essentials	0-596-00315-3	O'Reilly
C# in a Nutshell	0-596-00181-9	O'Reilly
COM and .Net Component Services	0-596-00103-7	O'Reilly
COM+ Programming with Visual Basic	1-565-92840-7	O'Reilly
Developing ASP Components	1-565-92750-8	O'Reilly
HTML & XHTML: The Definitive Guide	0-596-00026-X	O'Reilly

Discussion

When your primary aim is to organize and enhance the output beyond the confines of the`DataGrid` and `GridView` control's default tabular display, the `Repeater` control is a good choice because, unlike a `DataGrid` or `GridView`, it has associated templates that allow you to use almost any HTML to format the displayed data. It has the advantage of being relatively lightweight and easy to use. When using `Repeater`, however, you should know about a handful of nuances that can make life easier and, in one instance, enhance performance.

Example 2-4 shows one of the most common approaches to using the`Repeater` control, which is to place the `asp:Repeater` element in a table and use its `HeaderTemplate, ItemTemplate`, and `AlternatingItemTemplate` attributes to format the displayed data as rows in the table.

A `HeaderTemplate` is used to define the header row of the table. In this example, the header is straight HTML with a single table row and three columns.

An `ItemTemplate` formats the even-numbered rows of data, and an `AlternatingItemTemplate` formats the odd-numbered rows. For both templates, a single row in the table is defined with the same three columns defined in the header template. In each of the three columns, data-binding statements (described later) define the data to be placed in each of the columns. The only differences between `ItemTemplate` and `AlternatingItemTemplate` are the stylesheet classes used to output the rows. If you do not need to output the data using alternating styles, omit the`AlternatingItemTemplate` attribute.

The data from the database is bound to the cells in the templates using the`Eval` method. The Title field is placed in the first column, the ISBN field is placed in the second column, and the Publisher field is placed in the third column:

```
Eval("Title")
Eval("ISBN")
Eval("Publisher")
```

```
<table width="90%" align="center"
 class="tableWithBorder"
 style="border-collapse:collapse;">
```

In Version 1.x, the data-bind statements had the following form:

```
<%# DataBinder.Eval(Container.DataItem, <expression> %>
```

In Version 2.0, the syntax has been simplified to the following form:

```
<%# Eval(<expression>) %>
```

If the header is pure HTML with no data binding required, removing the `HeaderTemplate` attribute and placing the header HTML before the `asp:Repeater` tag will be more efficient. By moving the header outside of the `asp:Repeater` tag, the creation of several server-side controls is eliminated, which reduces the time required to render the page and improves performance.

```
<table width="90%" border="2" align="center"
   bordercolor="#000080" bgcolor="#FFFFE0"
   style="border-style:solid;border-collapse:collapse;">

 <thead bgcolor="#000080" class="TableHeader">
  <tr>
   <th align="center">Title</th>
   <th align="center">ISBN</th>
   <th align="center">Publisher</th>
  </tr>
 </thead>
```

The `Page_Load` method in the code-behind, shown in Examples 2-5 (VB) and 2-6 (C#), creates a `SqlDataSource`, sets the properties to define the source of the data and the select command to retrieve the data from the database, and binds the data source to the `GridView` control.

See Also

Recipe 2.6 for another example of using a template with a tabular control

More About the Eval Method

Recipe 2.3 uses the `Eval` method to bind data from the database to cells in the templates, as in:

```
<%#Eval("Title") %>
```

The advantage of this approach is its simple syntax. However, because the `Eval` method uses late-bound reflection to parse and evaluate a data-binding expression (in this case, it's a simple string), there's a performance penalty associated with it.

VB

```
<%#CType(Container.DataItem,DbDataRecord).Item("Title")%>
```

```
<%# ((DbDataRecord)Container.DataItem)["Title"] %>
```

For this syntax to work, explicit casting is required. When casting to DbDataRecord, add the following page-level directive to the beginning of your *.aspx* file:

```
<%@ Import namespace="System.Data.Common" %>
```

Example 2-4. Templates with Repeater control (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02TemplatesWithRepeaterVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02TemplatesWithRepeaterVB"
 Title="Templates with Repeater" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Templates With Repeater (VB)
 </div>
```

```
  <!-- Create a table within the cell to provide localized
     customization for the book list -->
  <table  width="90%"  align="center"
      class="tableWithBorder"
      style="border-collapse:collapse;">
   <asp:Repeater ID="repBooks" Runat="server">
    <HeaderTemplate>
     <thead class="tableHeader">
     <tr>
     <th align="center">
     Title</th>
     <th align="center">
     ISBN</th>
     <th align="center">
     Publisher</th>
     </tr>
     </thead>
    </HeaderTemplate>
    <ItemTemplate>
     <tr class="tableCellNormal">
     <td>
     <%#Eval("Title")%>
     </td>
     <td align="center">
     <%#Eval("ISBN")%>
     </td>
     <td align="center">
     <%#Eval("Publisher")%>
     </td>
     </tr>
    </ItemTemplate>
    <AlternatingItemTemplate>
     <tr class="tableCellAlternating">
     <td>
     <%#Eval("Title")%>
     </td>
     <td align="center">
     <%# Eval("ISBN") %>
     </td>
     <td align="center">
     <%#Eval("Publisher")%>
     </td>
     </tr>
    </AlternatingItemTemplate>
   </asp:Repeater>
  </table>
</asp:Content>
```

Example 2-5. Templates with Repeater control code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02TemplatesWithRepeaterVB.aspx
 ''' </summary>
 Partial Class CH02TemplatesWithRepeaterVB
  Inherits System.Web.UI.Page

  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim dSource As SqlDataSource = Nothing

   If (Not Page.IsPostBack) Then
    'configure the data source to get the data from the database
    dSource = New SqlDataSource()
    dSource.ConnectionString = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dSource.DataSourceMode = SqlDataSourceMode.DataReader
    dSource.ProviderName = "System.Data.OleDb"
    dSource.SelectCommand = "SELECT Title, ISBN, Publisher " & _
       "FROM Book " & _
       "ORDER BY Title"

    'set the source of the data for the repeater control and bind it
    repBooks.DataSource = dSource
    repBooks.DataBind()
   End If
  End Sub 'Page_Load
 End Class 'CH02TemplatesWithRepeaterVB
End Namespace
```

Example 2-6. Templates with Repeater control code-behind (.cs)

```
using System;
using System.Configuration;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02TemplatesWithRepeaterCS.aspx
 /// </summary>
 public partial class CH02TemplatesWithRepeaterCS : System.Web.UI.Page
 {

   ///************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   SqlDataSource dSource = null;

   if (!Page.IsPostBack)
   {

    //configure the data source to get the data from the database
    dSource = new SqlDataSource();
    dSource.ConnectionString = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dSource.DataSourceMode = SqlDataSourceMode.DataReader;
    dSource.ProviderName = "System.Data.OleDb";
    dSource.SelectCommand = "SELECT Title, ISBN, Publisher " +
       "FROM Book " +
       "ORDER BY Title";

    //set the source of the data for the repeater control and bind it
    repBooks.DataSource = dSource;
    repBooks.DataBind();
    }
  } // Page_Load
 } // CH02TemplatesWithRepeaterCS
}
```

Recipe 2.5. Displaying Data from an XML File

Problem

You want a quick and convenient way to display data from an XML file.

Solution

Use a `DataGrid` control and the `ReadXml` method of the `DataSet` class.

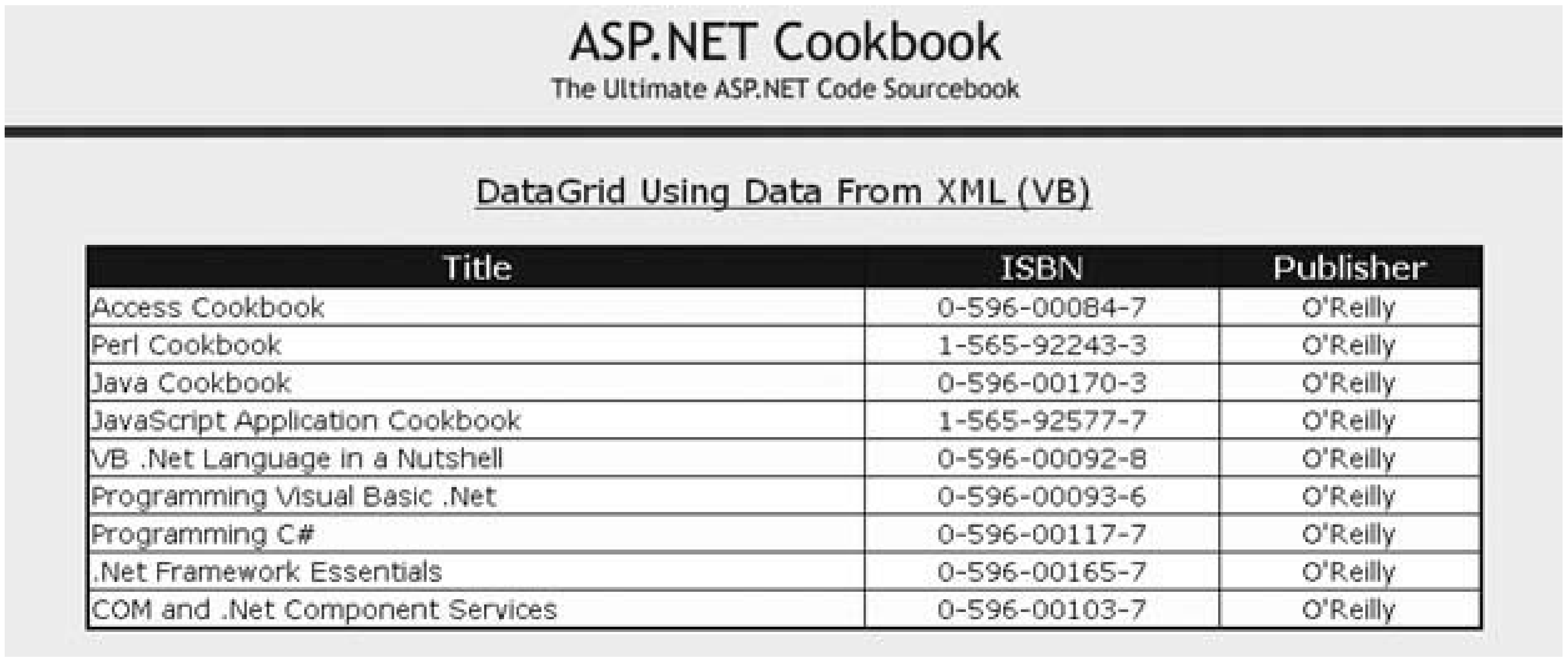
In the *.aspx* file, add a `DataGrid` control for displaying the data.

In the code-behind class for the page, use the .NET language of your choice to:

1. Read the data from the XML file using the `ReadXml` method of the `DataSet` class.
2. Bind the `DataSet` to the `DataGrid` control.

Figure 2-3 shows the appearance of a typical `DataGrid` in a browser. Example 2-7 shows the XML used for the recipe. Examples 2-8, 2-9 through 2-10 show the *.aspx* and code-behind files for an application that produces this result.

Figure 2-3. DataGrid with XML data output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

DataGrid Using Data From XML (VB)

Title	ISBN	Publisher
Access Cookbook	0-596-00084-7	O'Reilly
Perl Cookbook	1-565-92243-3	O'Reilly
Java Cookbook	0-596-00170-3	O'Reilly
JavaScript Application Cookbook	1-565-92577-7	O'Reilly
VB .Net Language in a Nutshell	0-596-00092-8	O'Reilly
Programming Visual Basic .Net	0-596-00093-6	O'Reilly
Programming C#	0-596-00117-7	O'Reilly
.Net Framework Essentials	0-596-00165-7	O'Reilly
COM and .Net Component Services	0-596-00103-7	O'Reilly

Discussion

The `Page_Load` method in the code-behind, shown in Examples 2-9 (VB) and 2-10 (C#), reads the data from the XML file using the `ReadXml` method of the `DataSet` class and then binds the `DataSet` to the `DataGrid` control.

Datasets are designed to support hierarchical data and can contain multiple tables of data. Because of this support, when data is loaded into the dataset, it is loaded into a Tables collection. In this example, a single node in the XML is called Book. The `DataSet` will automatically load the XML data into a table named Book. When binding the data to the `DataGrid`, you must reference the desired table if the `DataSet` contains more than one table. Reference the table by name instead of by index because the index value can change if the structure of the data changes.

The `DataGrid` control is one of the more flexible controls provided with ASP.NET. It outputs a complete HTML table with the bound data displayed in its cells. When used with a richdata source, such as a data reader, a `DataTable`, or a `DataSet`, the `DataGrid` can automatically generate columns for the data, complete with column headers (see Recipe 2.2 for an example using a `GridView` but in which you can easily substitute a `DataGrid` instead). Unfortunately, its default appearance and automatic behavior rarely meet the needs of a project. In this section, we discuss some ways to make changing the default appearance and behavior a little easier, especially as it relates to displaying XML data.

First, provide more flexibility for your graphical design team to achieve the desired appearance by defining an `asp:DataGrid` with `HeaderStyle`, `ItemStyle`, `AlternatingItemStyle`, and `Columns` elements, as shown in Example 2-8.

In this example, we use the `BorderColor` and the `BorderWidth` attributes of the `asp:DataGrid` element to define the color and width of the border around the table generated by the `DataGrid`. The `AutoGenerateColumns` attribute is set to `False` to allow us to define the columns that will be displayed in the grid. If this attribute is set to `true`, the `DataGrid` will automatically generate columns as a function of the data bound to the grid.

> In some cases, it is valid to have the `AutoGenerateColumns` attribute set to `true` *and*, as described in this recipe, define columns that will be displayed in the grid. This combined approach can be useful if you want to add another column to the ones automatically generated. When improperly done, it can result in duplicate columns.

The `HeaderStyle` element and its attributes are used to define the appearance of the grid header. The `ItemStyle` element and its attributes are used to define the appearance of the even-numbered rows in the grid. The `AlternatingItemStyle` element and its attributes are used to define the appearance of the odd-numbered rows in the grid. If you do not need to output the data using alternating styles, then omit the `AlternatingItemStyle` element.

The `Columns` element is used to define the columns in the grid, their headings, and the data fields bound to each of the columns. For each column appearing in the grid, an `asp:BoundColumn` element must be included. At a minimum, each `asp:BoundColumn` element must define the `HeaderTitle` attribute and the `DataField` attribute. The `HeaderTitle` attribute is set to the label for the column. The `DataField` attribute is set to the name of the data field in the dataset whose data is to be bound to the column. In addition, many other attributes can be included to define alignment, fonts, stylesheet classes, and the like as required to achieve the desired appearance.

See Also

Recipe 2.2

Example 2-7. XML data used for example

```
<Root>
  <Book>
   <BookID>1</BookID>
  <Title>Access Cookbook</Title>
   <ISBN>0-596-00084-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>2</BookID>
  <Title>ASP.NET Cookbook</Title>
   <ISBN>0-596-00378-1</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>3</BookID>
  <Title>Perl Cookbook</Title>
   <ISBN>1-565-92243-3</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
```

```
   <BookID>4</BookID>
   <Title>Java Cookbook</Title>
   <ISBN>0-596-00170-3</ISBN>
   <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>5</BookID>
   <Title>JavaScript Application Cookbook</Title>
   <ISBN>1-565-92577-7</ISBN>
   <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>6</BookID>
   <Title>VB .Net Language in a Nutshell</Title>
   <ISBN>0-596-00092-8</ISBN>
   <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>7</BookID>
   <Title>Programming Visual Basic .Net</Title>
   <ISBN>0-596-00093-6</ISBN>
   <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>8</BookID>
   <Title>Programming C#</Title>
   <ISBN>0-596-00117-7</ISBN>
   <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>9</BookID>
   <Title>.Net Framework Essentials</Title>
   <ISBN>0-596-00165-7</ISBN>
   <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>10</BookID>
   <Title>COM and .Net Component Services</Title>
   <ISBN>0-596-00103-7</ISBN>
   <Publisher>O'Reilly</Publisher>
 </Book>
</Root>
```

Example 2-8. DataGrid with XML data (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02DataGridWithXMLVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02DataGridWithXMLVB"
 Title="Datagrid With XML" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  DataGrid Using Data From XML (VB)
 </div>
 <asp:DataGrid ID="dgBooks" Runat="server"
     BorderColor="#000080" BorderWidth="2px"
     AutoGenerateColumns="False"
     Width="90%" HorizontalAlign="Center" >
  <HeaderStyle HorizontalAlign="Center"
     CssClass="tableHeader" />
  <ItemStyle CssClass="tableCellNormal" />
  <AlternatingItemStyle CssClass="tableCellAlternating" />
  <Columns>
   <asp:BoundColumn HeaderText="Title"
     DataField="Title" />
   <asp:BoundColumn HeaderText="ISBN"
     DataField="ISBN"
     ItemStyle-HorizontalAlign="Center" />
   <asp:BoundColumn HeaderText="Publisher"
     DataField="Publisher"
     ItemStyle-HorizontalAlign="Center" />
  </Columns>
 </asp:DataGrid>
</asp:Content>
```

Example 2-9. DataGrid with XML data code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
  '''  CH02DataGridWithXMLVB.aspx
 ''' </summary>
 Partial  Class  CH02DataGridWithXMLVB
  Inherits System.Web.UI.Page

   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Const BOOK_TABLE As String = "Book"

   Dim dSet As DataSet = Nothing
   Dim xmlFilename As String

   If (Not Page.IsPostBack) Then
    'get fully qualified path to the "books" xml document located
    'in the xml directory
    xmlFilename = Server.MapPath("xml") & "\books.xml"

    'create a dataset and load the books xml document into it
    dSet = New DataSet
    dSet.ReadXml(xmlFilename)

    'bind the dataset to the datagrid
    dgBooks.DataSource = dSet.Tables(BOOK_TABLE)
    dgBooks.DataBind()

   End If
  End Sub 'Page_Load
 End  Class  'CH02DataGridWithXMLVB
End Namespace
```

Example 2-10. DataGrid with XML data code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02DataGridWithXMLCS.aspx
 /// </summary>
 public partial class CH02DataGridWithXMLCS : System.Web.UI.Page
 {
  ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   const String BOOK_TABLE = "Book";

   DataSet dSet = null;
   String xmlFilename;

   if (!Page.IsPostBack)
   {
    // get fully qualified path to the "books" xml document located
    // in the xml directory
    xmlFilename = Server.MapPath("xml") + "\\books.xml";

    // create a dataset and load the books xml document into it
    dSet = new DataSet();
    dSet.ReadXml(xmlFilename);

    // bind the dataset to the datagrid
    dgBooks.DataSource = dSet.Tables[BOOK_TABLE];
    dgBooks.DataBind();

   }
  } // Page_Load
 } // CH02DataGridWithXMLCS
}
```

PREV

Recipe 2.6. Displaying an Array as a Group of Checkboxes

Problem

You have data in an array that needs to be displayed as a group of checkboxes.

Solution

Use a `CheckBoxList` control and bind the array to it.

Add a `CheckBoxList` control to the *.aspx* file.

In the code-behind class for the page, bind the array to the`CheckBoxList` control.

Figure 2-4 shows the appearance of a typical `CheckBoxList` in a browser, with a couple of checkboxes preselected. Examples 2-11, 2-12 through 2-13 show the *.aspx* and code-behind files for an application that produces this result.

Figure 2-4. CheckBoxList with array data output

Discussion

The `CheckBoxList` control simplifies the job of generating a list ofcheckboxes. Here's a rundown of some of the attributes that control the checkbox display. In the example that we developed for this recipe, we have placed a `CheckBoxList` control in a Table cell to control its position on the form as shown in Example 2-11.

The RepeatColumns attribute of the CheckBoxList control is used to set the number of columns in which the checkboxes are to be displayed.

The RepeatDirection attribute is set to Horizontal, which displays the checkboxes in rows from left to right and then top to bottom. This attribute can also be set toVertical to display the checkboxes in columns from top to bottom and then left to right.

The RepeatLayout attribute is set to Table, which causes the CheckBoxList control to output an HTML table that contains the checkboxes. Using Table ensures the checkboxes are aligned vertically. This attribute can be set to Flow, which causes the CheckBoxList control to output a <span> element for the checkboxes with <br> elements, thus placing the checkboxes in rows. In this case, unless all of your data is the same size, the checkboxes will not be aligned vertically.

The CssClass attribute controls the format of the text displayed with the checkboxes, and thewidth attribute sets the width of the generated HTML table.

The styles attribute can be used to format the data in any manner supported by inline HTML styles. If you're considering using inline styles though, remember that some older browsers do not fully support them.

If you need a list of radio buttons instead of checkboxes, substitute RadioButtonList for CheckBoxList in the *.aspx* file.

The Page_Load method in the code-behind, shown in Examples 2-12 (VB) and 2-13 (C#), builds the array of data by declaring an ArrayList and adding the text for the checkboxes to the ArrayList. It then sets the source of the data to the ArrayList and performs a data bind.

The ArrayList class provides a convenient, lightweight container for data that is to be bound to list controls. An ArrayList is almost identical to the built-in Array type but adds automatic size management that makes it easier to use.

To preselect a single checkbox, set the SelectedIndex property of the CheckBoxList control to the index of the item that is to be preselected. If multiple checkboxes need to be preselected, use the FindByText (or FindByValue) method of the Items collection in the CheckBoxList control to find the appropriate item(s), and then set the Selected property to true, as shown in Examples 2-12 (VB) and 2-13 (C#).

See Also

The ArrayList class in the MSDN Library

Example 2-11. CheckBoxList with array data (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02CheckboxListWithArrayVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02CheckboxListWithArrayVB"
 Title="Checkbox List With Array" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  CheckBoxList With Array (VB)
 </div>
 <div  style="margin-left:50px">
  <asp:CheckBoxList id="cbBooks" runat="server"
     RepeatColumns="2"
     RepeatDirection="Horizontal"
     RepeatLayout="Table"
     CssClass="MenuItem"
     TextAlign="Right"
     width="100%" />
 </div>
</asp:Content>
```

Example 2-12. CheckBoxList with array data code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02CheckboxListWithArrayVB.aspx
 ''' </summary>
 Partial  Class  CH02CheckboxListWithArrayVB
  Inherits System.Web.UI.Page

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim values As ArrayList

   If (Not Page.IsPostBack) Then
    'build array of data to bind to checkboxlist
    values = New ArrayList
    values.Add("Access Cookbook")
    values.Add("ASP.NET Cookbook")
    values.Add("Perl Cookbook")
    values.Add("Java Cookbook")
    values.Add("VB .Net Language in a Nutshell")
    values.Add("Programming Visual Basic .Net")
    values.Add("Programming C#")
    values.Add(".Net Framework Essentials")
    values.Add("COM and .Net Component Services")

    'bind the data to the checkboxlist
    cbBooks.DataSource = values
    cbBooks.DataBind()

    'preselect several books
    cbBooks.Items.FindByText("ASP.NET Cookbook").Selected = True
    cbBooks.Items.FindByText("Programming C#").Selected = True
   End If
  End Sub 'Page_Load
 End Class 'CH02CheckboxListWithArrayVB
End Namespace
```

Example 2-13. CheckBoxList with array data code-behind (.cs)

```
using System;
using System.Collections;
using System.Configuration;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02CheckboxListWithArrayCS.aspx
 /// </summary>
 public partial class CH02CheckboxListWithArrayCS : System.Web.UI.Page
 {
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   ArrayList values;

   if (!Page.IsPostBack)
   {

    // build array of data to bind to checkboxlist
    values = new ArrayList();
    values.Add("Access Cookbook");
    values.Add("ASP.NET Cookbook");
    values.Add("Perl Cookbook");
    values.Add("Java Cookbook");
    values.Add("VB .Net Language in a Nutshell");
    values.Add("Programming Visual Basic .Net");
    values.Add("Programming C#");
    values.Add(".Net Framework Essentials");
    values.Add("COM and .Net Component Services");

    // bind the data to the checkboxlist
    cbBooks.DataSource = values;
    cbBooks.DataBind();
    // preselect several books
    cbBooks.Items.FindByText("ASP.NET Cookbook").Selected = true;
    cbBooks.Items.FindByText("Programming C#").Selected = true;
    }
  } // Page_Load
 }  //  CH02CheckboxListWithArrayCS
}
```

PREV

Recipe 2.7. Displaying Data from a Hashtable

Problem

You have data in a Hashtable , a class that provides the ability to store a collection ofkey/value pairs, and you want to display the data in a columnar table.

Solution

Use a DataList control and bind the Hashtable to it.

Add a DataList control to the *.aspx* file, being careful to place it in a Table cell to control its position on the form.

In the code-behind class for the page, use the .NET language of your choice to:

1. Define the Hashtable as the data source for the DataList control.
2. Set the control's key and value.
3. Bind the Hashtable to the DataList control.

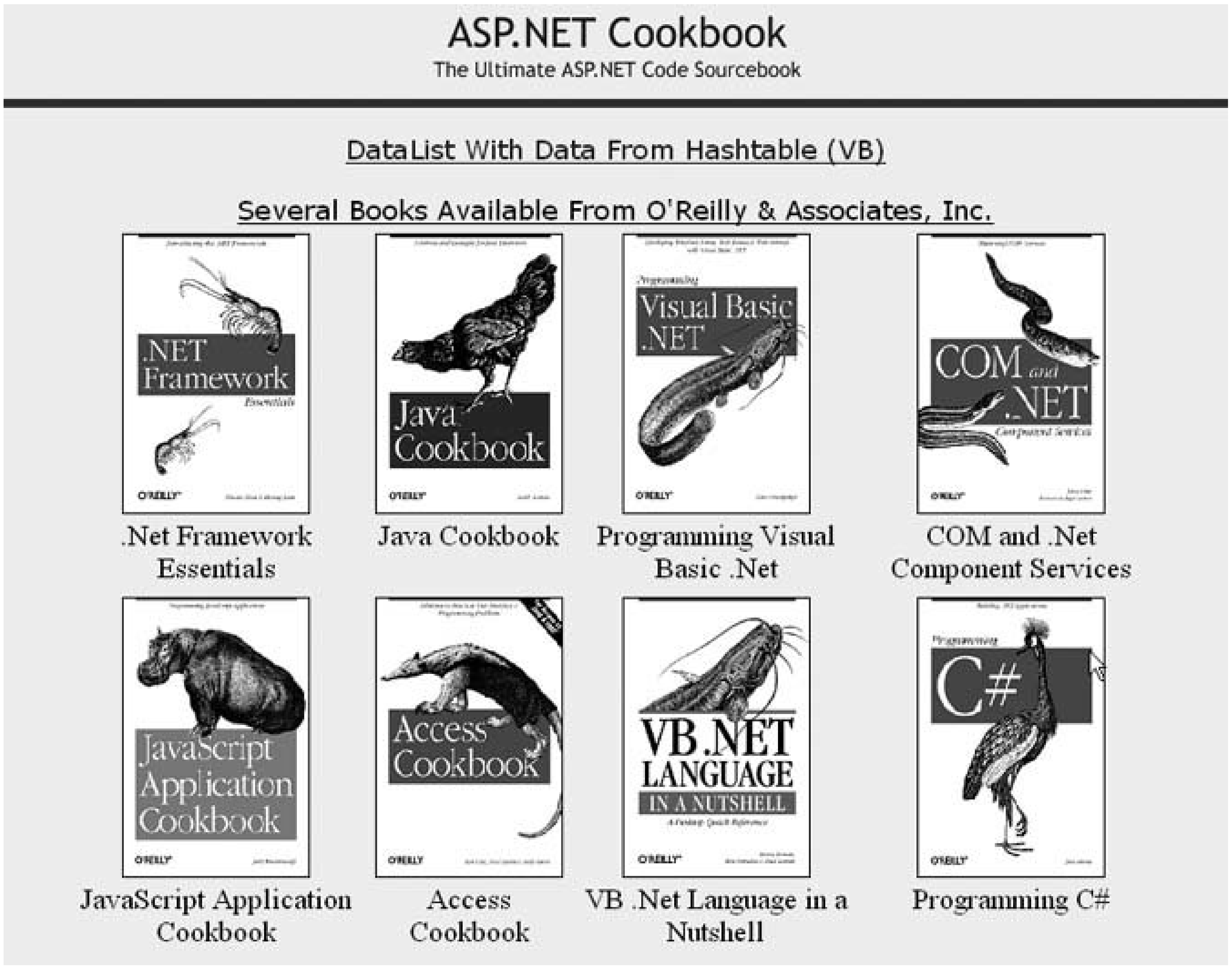
Figure 2-5 shows the appearance of a typical DataList within a browser that has been bound to a Hashtable filled with, in our case, book data. Examples 2-14, 2-15 through 2-16 show the *.aspx* and code-behind files for an application that produces this result.

Discussion

The DataList control can display almost any data type in various ways using its available templates and styles. Templates are available for the header, footer, items, alternating items, separators, selected items, and edit items to define and organize the data to output. Styles are available for eacl of the templates to define how the content appears.

In this example, an asp:DataList control is placed in a Table cell to control its position on the form, a shown in Example 2-14 . The RepeatColumns attribute of the control defines the number of columns that should be output, which in this case is 4.

Figure 2-5. DataList with Hashtable data output


ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

DataList With Data From Hashtable (VB)

Several Books Available From O'Reilly & Associates, Inc.

.Net Framework Essentials	Java Cookbook	Programming Visual Basic .Net	COM and .Net Component Services
JavaScript Application Cookbook	Access Cookbook	VB .Net Language in a Nutshell	Programming C#

The RepeatDirection attribute indicates that the data should be output horizontally, which displays the data in rows from left to right and then top to bottom. TheRepeatLayout attribute indicates the data should be output in an HTML table, which provides the greatest flexibility in arranging the data items.

The HeaderTemplate element defines a simple line of text to be used as a header for the data list. The HeaderTemplate can contain any HTML and ASP.NET controls.

The HeaderStyle element defines the positioning of the header as well as the stylesheet class used to define the text formatting. A large number of style attributes are available to format the header data

The ItemTemplate element defines an HTML table that contains an image and text describing the image. A table controls the positioning of the data items.

The ItemStyle element defines the positioning of the items output using theItemTemplate . In this example, only horizontal and vertical positioning are defined, but many styles are available.

The Page_Load method of the code-behind, shown in Examples 2-15(VB) and 2-16 (C#), uses a Hashtable as the container for the data to bind to the DataList . A Hashtable provides the ability to store a collection of key/value pairs. This is the equivalent of a two-column table, which provides a lightweight container for data when only two items are required per row, such as in this example.

This example builds the `Hashtable` of data by declaring a `Hashtable` object and adding the key/value pairs to the `Hashtable` . It then sets the source of the data to the `Hashtable` object, defines the key field (the key in the `Hashtable`) and the data member (the value in the `Hashtable`), and performs a data bind.

See Also

The `DataList` class and the `HashTable` class in the MSDN Library

Example 2-14. DataList with Hashtable data (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02DataListWithHashtableVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02DataListWithHashtableVB"
 Title="DataList With Hashtable" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">
  DataList With Data From Hashtable (VB)
 </div>
 <asp:DataList id="dlBooks" runat="server"
   RepeatColumns="4"
   RepeatDirection="Horizontal"
   RepeatLayout="Table"
   Caption="Several Books Available From O'Reilly & Associates, Inc.">
 <HeaderStyle HorizontalAlign="Center"
    CssClass="BlackPageHeading" />
 <ItemStyle HorizontalAlign="Center" VerticalAlign="Top" />
 <ItemTemplate>
  <table align="center">
   <tr>
    <td align="center">
    <img src="<%#Eval("Value")%>"
    height="145" >
    alt="BookImage"/>
    </td>
   </tr>
   <tr>
    <td align="center">
    <%#Eval("Key")%>
    </td>
   </tr>
   </table>
  </ItemTemplate>
 </asp:DataList>
</asp:Content>
```

Example 2-15. DataList with Hashtable data code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02DataListWithHashtableVB.aspx
 ''' </summary>
 Partial Class CH02DataListWithHashtableVB
  Inherits System.Web.UI.Page

   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim values As Hashtable

   If (Not Page.IsPostBack) Then
    'build HashTable with the names of the books as the key and the
    'relative path to the cover image as the value
    values = New Hashtable
    values.Add(".Net Framework Essentials", _
       "images/books/DotNetFrameworkEssentials.gif")
    values.Add("Access Cookbook", _
       "images/books/AccessCookbook.gif")
    values.Add("ASP.NET Cookbook", _
       "images/books/ASPNetCookbook.gif")
    values.Add("Java Cookbook", _
       "images/books/JavaCookbook.gif")
    values.Add("JavaScript Application Cookbook", _
       "images/books/JavaScriptCookbook.gif")
    values.Add("Programming C#", _
       "images/books/ProgrammingCSharp.gif")
    values.Add("Programming Visual Basic .Net", _
       "images/books/ProgrammingVBDotNet.gif")
    values.Add("VB .Net Language in a Nutshell", _
       "images/books/VBDotNetInANutshell.gif")
    'define the data source, key, value, and bind to the Hashtable
    dlBooks.DataSource = values
    dlBooks.DataKeyField = "Key"
    dlBooks.DataMember = "Value"
    dlBooks.DataBind()
   End If
  End Sub 'Page_Load
 End Class 'CH02DataListWithHashtableVB
End Namespace
```

Example 2-16. DataList with Hashtable data code-behind (.cs)

```
using System;
using System.Collections;
using System.Configuration;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02DataListWithHashtableCS.aspx
 /// </summary>
 public partial class CH02DataListWithHashtableCS : System.Web.UI.Page
 {
  ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   Hashtable values;

   if (!Page.IsPostBack)
   {
    // build HashTable with the names of the books as the key and the
    // relative path to the cover image as the value
    values = new Hashtable();
    values.Add(".Net Framework Essentials",
    "images/books/DotNetFrameworkEssentials.gif");
    values.Add("Access Cookbook",
    "images/books/AccessCookbook.gif");
    values.Add("ASP.NET Cookbook",
    "images/books/ASPNetCookbook.gif");
    values.Add("Java Cookbook",
    "images/books/JavaCookbook.gif");
    values.Add("JavaScript Application Cookbook",
    "images/books/JavaScriptCookbook.gif");
    values.Add("Programming C#",
    "images/books/ProgrammingCSharp.gif");
    values.Add("Programming Visual Basic .Net",
    "images/books/ProgrammingVBDotNet.gif");
    values.Add("VB .Net Language in a Nutshell",
    "images/books/VBDotNetInANutshell.gif");

    // define the data source, key, value, and bind to the Hashtable
    dlBooks.DataSource = values;
    dlBooks.DataKeyField = "Key";
    dlBooks.DataMember = "Value";
    dlBooks.DataBind();
   }
  } // Page_Load
 } //  CH02DataListWithHashtableCS
}
```

PREV

Recipe 2.8. Adding Next/Previous Navigation to a DataGrid

Problem

You need to display data from a database in a table; the database has more rows than can fit on a single page, so you want to use next/previous buttons for navigation.

Solution

Use a `DataGrid` control, enable its built-in pagination features, and then bind the data to it.

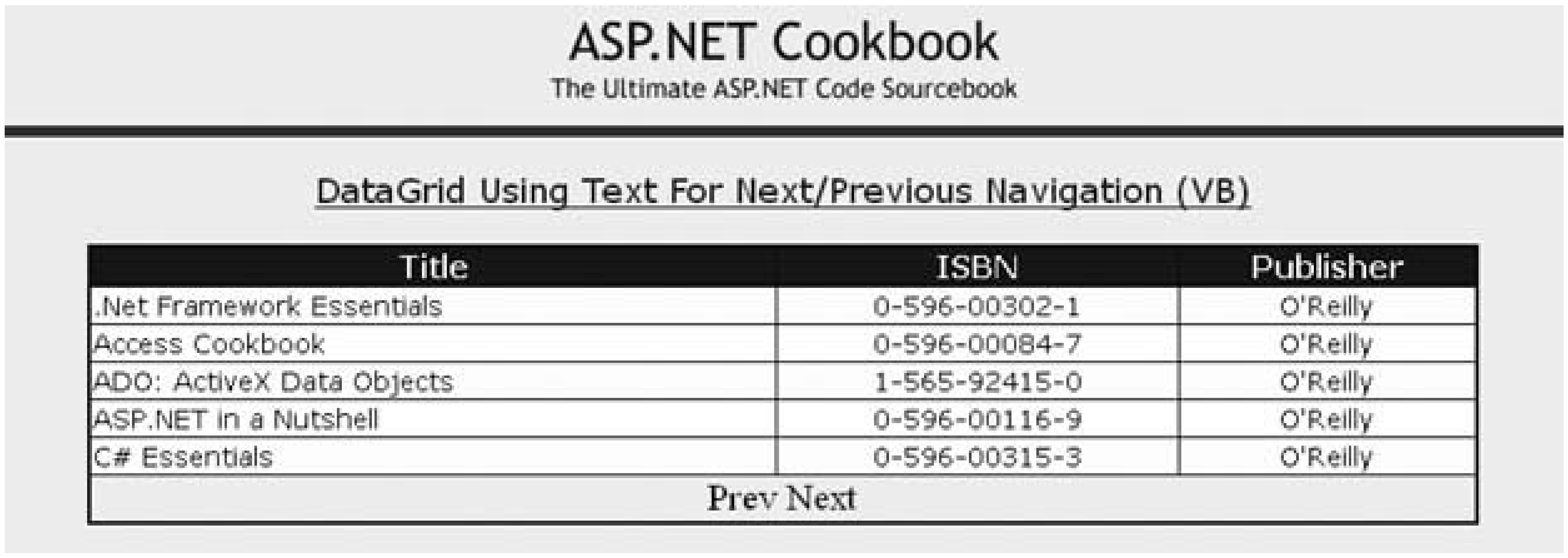
Add a `DataGrid` control to the *.aspx* file, and use its `AllowPaging` and other related attributes to enable pagination.

In the code-behind class for the page, use the .NET language of your choice to:

1. Create a routine that binds a `DataSet` to the `DataGrid` in the usual fashion.
2. Create an event handler that performs the page navigation, for example, one that handles the `PageIndexChanged` event for the `DataGrid` and rebinds the data.

Figure 2-6 shows the appearance of a typical `DataGrid` within a browser with next/previous navigation. Examples 2-17, 2-18 through 2-19 show the *.aspx* and code-behind files for an application that produces this result.

Figure 2-6. DataGrid with next/previous navigation output


ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

DataGrid Using Text For Next/Previous Navigation (VB)

Title	ISBN	Publisher
.Net Framework Essentials	0-596-00302-1	O'Reilly
Access Cookbook	0-596-00084-7	O'Reilly
ADO: ActiveX Data Objects	1-565-92415-0	O'Reilly
ASP.NET in a Nutshell	0-596-00116-9	O'Reilly
C# Essentials	0-596-00315-3	O'Reilly

Prev Next

Discussion

The DataGrid control includes the ability to perform pagination of the data that is displayed in the grid; using the built-in pagination requires little code. Pagination is enabled and configured by the attributes of the DataGrid element:

```
AllowPaging="True"
PageSize="5"
PagerStyle-Mode="NextPrev"
PagerStyle-Position="Bottom"
PagerStyle-HorizontalAlign="Center"
PagerStyle-NextPageText="Next"
PagerStyle-PrevPageText="Prev"
```

Setting the AllowPaging attribute to TRue enables paging for the DataGrid, and the PageSize attribute defines the number of rows that will be displayed in a single page. Setting thePageStyle-Mode attribute to NextPrev enables the output of the Next/Prev controls (see Recipe 2.9 for other uses of this attribute).

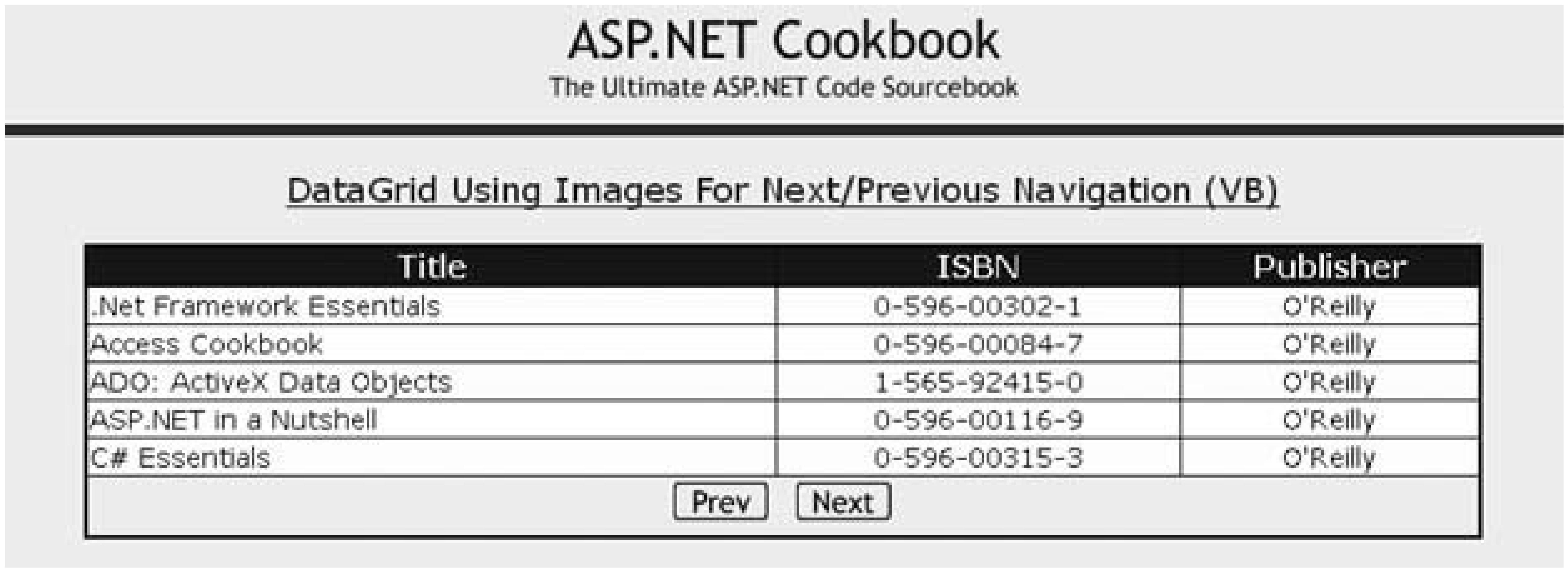
The remaining attributes define how the pagination controls look.PagerStyle-Position defines the location of the Next/Prev controls. Valid values include Bottom, Top, and TopAndBottom. PagerStyle-HorizontalAlign defines the horizontal positioning of the Next/Prev controls. Valid values include Left, Center, Right, and NotSet. NotSet is effectively the same as Left because Left is the default.

PagerStyle-NextPageText defines the text to output for the next page navigation control, and PagerStyle-PrevPageText defines the text to output for the previous page navigation.

The PagerStyle-NextPageText and PagerStyle-PrevPageText attribute values can include HTML to format the text of the controls. Almost any HTML can be used, including image tags. If you change the values of the two text attributes, the Next/Prev controls will be output as shown in Figure 2-7.

```
PagerStyle-NextPageText=
  "<img src='images/button_next.gif' border='0'>"
PagerStyle-PrevPageText=
  "<img src='images/button_prev.gif' border='0'>">
```

Figure 2-7. DataGrid output using image tags for next/previous controls



ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

DataGrid Using Images For Next/Previous Navigation (VB)

Title	ISBN	Publisher
.Net Framework Essentials	0-596-00302-1	O'Reilly
Access Cookbook	0-596-00084-7	O'Reilly
ADO: ActiveX Data Objects	1-565-92415-0	O'Reilly
ASP.NET in a Nutshell	0-596-00116-9	O'Reilly
C# Essentials	0-596-00315-3	O'Reilly

The bindData routine, shown in the code-behind in Examples 2-18 (VB) and 2-19 (C#), performs the data binding. This routine provides the typical binding of a dataset to the DataGrid. No additional code is required in this routine to support the default pagination.

The SqlDataSource and ObjectDataSource data source controls provided in ASP.NET 2.0 cannot be used with a DataGrid when paging is implemented. The DataGrid requires its data source to implement ICollection to support paging. The ICollection interface is not implemented by the new data source controls. Using a data source control with a DataGrid when pagination is implemented will result in an exception such as the following being thrown:

> AllowCustomPaging must be TRue and VirtualItemCount must be set for a DataGrid with ID "dgBooks" when AllowPaging is set to true and the selected data source does not implement ICollection.

The dgBooks_PageIndexChanged event handler provides the code required to perform the page navigation. The new page number to display is passed in the event arguments (e). The `CurrentPageIndex` property of the `DataGrid` must be set to the passed value, and the data must be rebound to the `DataGrid`.

> The default pagination code shown in this recipe can be inefficient when used with data containing a large number of rows. By default, all of the data for a query is returned and used to populate the `DataSet`. When the query returns a small set of data (fewer than 100 rows and a small number of columns), the pagination shown in this recipe is adequate for most applications. If your query returns a million rows, the performance of your application will be unacceptable. See Recipe 2.12 for a more efficient approach to the pagination of large datasets.

Event Handlers

The method used to connect event handlers to events in Version 1.x of ASP.NET was implemented in different ways for VB and C#. VB required two additions to your code to handle an event. First, the control that had events to handle needed to be declared in the code-behind class using the `WithEvents` keyword as shown here:

```
Protected WithEvents dgBooks As DataGrid
```

Second, the method used to handle an event needed to have the `Handles` keyword added to the end of the method and include the control and event to be handled. This informed the compiler to add the code required to "wire" the event to the method.

```
Private Sub dgBooks_PageIndexChanged(ByVal source As Object,
 ByVal e As DataGridPageChangedEventArgs)
Handles dgBooks.PageIndexChanged
```

In C#, events were connected to methods by explicitly creating a new event handler of the required type and "wiring" it to the desired control's event:

```
this.dgBooks.PageIndexChanged +=
 new DataGridPageChangedEventHandler (this.dgBooks_PageIndexChanged);
```

Version 2.0 of ASP.NET has implemented a simpler and more consistent method of connecting event handlers to events. Server controls now have new attributes to define the server-side event handlers to be called when a specific event occurs. For example, the `DataGrid` has an `OnPageIndexChanged` attribute used in this recipe to connect the `OnPageIndexChanged` event to the `dgBooks_PageIndexChanged` event handler in the code-behind class:

```
 <asp:DataGrid
  id="dgBooks"
  runat="server"

  …

  OnPageIndexChanged="dgBooks_PageIndexChanged" >

  …

 </asp:DataGrid>
```

To provide backward compatibility, the approach used in Version 1.x can be used in Version 2.0.

See Also

Recipes 2.8, 2.9, and 2.12 for other examples of pagination

Example 2-17. DataGrid with next/previous navigation (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH02DatagridWithNextPrevNavVB1.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH02DatagridWithNextPrevNavVB1"
  Title="Datagrid With Text For Next/Prev Navigation" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  DataGrid Using Text For Next/Previous Navigation (VB)
 </div>
 <asp:DataGrid
  id="dgBooks"
  runat="server"
  BorderColor="000080"
  BorderWidth="2px"
  AutoGenerateColumns="False"
  HorizontalAlign="Center"
  Width="90%"
  AllowPaging="True"
  PageSize="5"
  PagerStyle-Mode="NextPrev"
  PagerStyle-Position="Bottom"
  PagerStyle-HorizontalAlign="Center"
  PagerStyle-NextPageText="Next"
  PagerStyle-PrevPageText="Prev"
  OnPageIndexChanged="dgBooks_PageIndexChanged" >
  <HeaderStyle HorizontalAlign="Center"
    CssClass="tableHeader" />
  <ItemStyle cssClass="tableCellNormal" />
  <AlternatingItemStyle cssClass="tableCellAlternating" />
  <Columns>
   <asp:BoundColumn HeaderText="Title" DataField="Title" />
   <asp:BoundColumn HeaderText="ISBN" DataField="ISBN"
     ItemStyle-HorizontalAlign="Center" />
   <asp:BoundColumn HeaderText="Publisher" DataField="Publisher"
     ItemStyle-HorizontalAlign="Center" />
  </Columns>
 </asp:DataGrid>
</asp:Content>
```

Example 2-18. DataGrid with next/previous navigation code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb
Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02DatagridWithNextPrevNavVB1.aspx
 ''' </summary>
 Partial  Class  CH02DatagridWithNextPrevNavVB1
  Inherits System.Web.UI.Page

  '''**********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
   If (Not Page.IsPostBack) Then
    bindData()
   End If
  End Sub 'Page_Load

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page index changed
  ''' event of the datagrid. It is responsible for setting the page index
  ''' from the passed arguments and rebinding the data.
  ''' </summary>
  '''
  ''' <param name="source">Set to the source of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub dgBooks_PageIndexChanged(ByVal source As Object, _
    ByVal e As System.Web.UI.WebControls.DataGridPageChangedEventArgs)
   'set new page index and rebind the data
   dgBooks.CurrentPageIndex = e.NewPageIndex
   bindData()
  End Sub 'dgBooks_PageIndexChanged

  '''*************************************************************************
  ''' <summary>
  ''' This routine queries the database for the data to displayed and binds
  ''' it to the datagrid
  ''' </summary>
  Private Sub bindData()
   Dim dbConn As OleDbConnection = Nothing
   Dim da As OleDbDataAdapter = Nothing
   Dim dSet As DataSet = Nothing
   Dim strConnection As String
   Dim strSQL As String
   Try
    'get the connection string from web.config and open a connection
    'to the database
    strConnection = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDbConnection(strConnection)
    dbConn.Open()

    'build the query string and get the data from the database
    strSQL = "SELECT Title, ISBN, Publisher " & _
     "FROM Book " & _
     "ORDER BY Title"
    da = New OleDbDataAdapter(strSQL, dbConn)
    dSet = New DataSet
    da.Fill(dSet)

    'set the source of the data for the datagrid control and bind it
    dgBooks.DataSource = dSet
    dgBooks.DataBind()
```

```
    Finally
     'cleanup
     If (Not IsNothing(dbConn)) Then
     dbConn.Close()
     End If
    End Try
  End Sub 'bindData
 End  Class  'CH02DatagridWithNextPrevNavVB1
End Namespace
```

Example 2-19. DataGrid with next/previous navigation code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
   ///  CH02DatagridWithNextPrevNavCS1.aspx
  /// </summary>
   public partial class CH02DatagridWithNextPrevNavCS1  :  System.Web.UI.Page
  {
   ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
    if (!Page.IsPostBack)
    {
     bindData();
    }
  } // Page_Load
   ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page index changed
  /// event of the datagrid. It is responsible for setting the page index
  /// from the passed arguments and rebinding the data.
  /// </summary>
  ///
  /// <param name="source">Set to the source of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void dgBooks_PageIndexChanged(Object source,
     System.Web.UI.WebControls.DataGridPageChangedEventArgs e)
  {
    // set new page index and rebind the data
    dgBooks.CurrentPageIndex = e.NewPageIndex;
    bindData();
  } // dgBooks_PageIndexChanged

   ///***************************************************************************
  /// <summary>
  /// This routine queries the database for the data to displayed and binds
  /// it to the datagrid
  /// </summary>
  private void bindData()
  {
    OleDbConnection dbConn = null;
    OleDbDataAdapter da = null;
    DataSet dSet = null;
    String strConnection;
    String strSQL;

    try
    {
     // get the connection string from web.config and open a connection
     // to the database
     strConnection = ConfigurationManager.
     ConnectionStrings["dbConnectionString"].ConnectionString;
     dbConn = new OleDbConnection(strConnection);
     dbConn.Open();

     // build the query string and get the data from the database
     strSQL = "SELECT Title, ISBN, Publisher " +
      "FROM Book " +
      "ORDER BY Title";
     da = new OleDbDataAdapter(strSQL, dbConn);
     dSet = new DataSet();
     da.Fill(dSet);

     // set the source of the data for the datagrid control and bind it
     dgBooks.DataSource = dSet;
     dgBooks.DataBind();
    }

    finally
    {
     // cleanup
     if (dbConn != null)
     {
     dbConn.Close();
     }
    }
   } // bindData
  } //  CH02DatagridWithNextPrevNavCS1
  }
```

PREV

PREV

Recipe 2.9. Adding First/Last Navigation to a DataGrid

Problem

You need to display data from a database in a table, but the database has more rows than will fit on a single page, and you want to use first/last buttons along with next/previous buttons for navigation.

Solution

Use a `DataGrid` control, add first/last and next/previous buttons (with event handlers for each one), and then bind the data to it.

In the *.aspx* file:

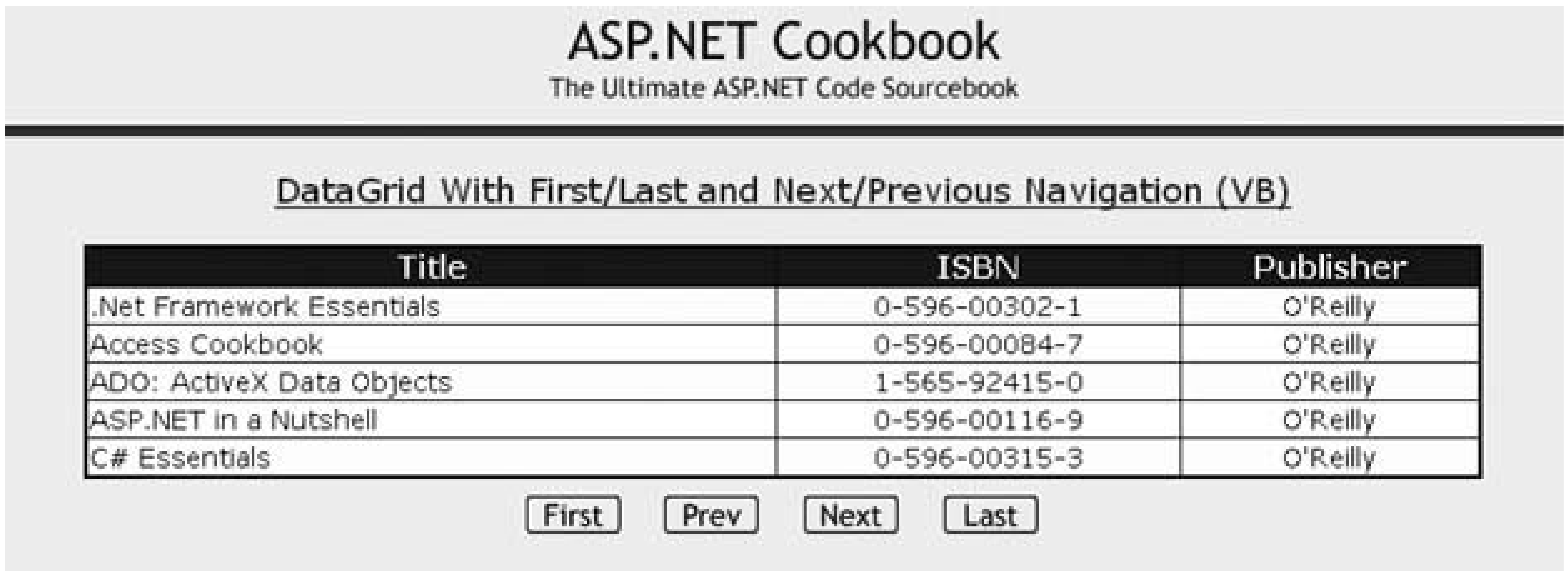
1. Add a `DataGrid` control to the *.aspx* file.
2. Add a row below the `DataGrid` with first/last and next/previous buttons for navigation.

In the code-behind class for the page, use the .NET language of your choice to:

1. Create a routine that binds a dataset to the `DataGrid` in the usual fashion.
2. For each of the four buttons, create an event handler to handle the button's click event, perforn the requisite page navigation, and rebind the data.

Figure 2-8 shows the appearance of a typical `DataGrid` within a browser with first/last and next/previous buttons for navigation. Examples 2-20, 2-21 through 2-22 show the *.aspx* and code-behind files for an application that produces this result.

Figure 2-8. DataGrid with first/last and next/previous navigation output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

DataGrid With First/Last and Next/Previous Navigation (VB)

Title	ISBN	Publisher
.Net Framework Essentials	0-596-00302-1	O'Reilly
Access Cookbook	0-596-00084-7	O'Reilly
ADO: ActiveX Data Objects	1-565-92415-0	O'Reilly
ASP.NET in a Nutshell	0-596-00116-9	O'Reilly
C# Essentials	0-596-00315-3	O'Reilly

Discussion

The main theme of this recipe is to provide an alternative to the `DataGrid` control's default pagination controls and, at the same time, handle the custom paging. Setting the `PagerStyle-Visible` attribute to `False` makes the pager invisible in a `DataGrid` control, allowing you to implement your own user interface for the pagination controls. (The *pager* is the element on the `DataGrid` control that allows you to link to other pages when paging is enabled.) When the pager is invisible, some appearance-related attributes for the pager are not required and can be eliminated, specifically `PagerStyle-Position`, `PagerStyle-HorizontalAlign`, `PagerStyle-NextPageText`, and `PagerStyle-PrevPageText`. Adding a row below the `DataGrid` to hold the four navigation buttons (Next, Prev, First, and Last) is a key ingredient.

In the application we have developed for this recipe, we added four event handler routines to the code-behind to handle the click events for the four buttons, a handy strategy for your application as well. The event handlers alter the current page index for the grid (`CurrentPageIndex`), as appropriate, and rebind the data.

To improve performance, the event handlers check to see if the page needs changing and rebinding prior to changing the current page index value. For example, the `btnPrev_ServerClick` handler checks to see if `CurrentPageIndex` is greater than zero before subtracting one from it.

To improve performance still further, you could add the following code to the end of the `bindData` method to disable the appropriate buttons when no action would be taken on the server callfor example, disabling the First and Prev buttons when the first page is displayed. This would avoid an unnecessary trip to the server.

```
Dim pageIndex as Integer = dgBooks.CurrentPageIndex
If (pageIndex = 0) Then
 btnFirst.Disabled = True
Else
 btnFirst.Disabled = False
End If
```

```
        If (pageIndex = dgBooks.PageCount - 1) then
         btnLast.Disabled = True
        Else
         btnLast.Disabled = False
        End If
```

See Also

Recipe 2.7 and the sidebar "Event Handlers" in Recipe 2.7

Example 2-20. DataGrid with first/last and next/previous navigation (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02DatagridWithFirstLastNavVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02DatagridWithFirstLastNavVB"
 Title="Datagrid With First/Last and Next/Prev Navigation" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  DataGrid With First/Last and Next/Previous Navigation (VB)
 </div>
 <asp:DataGrid ID="dgBooks" Runat="server"
     BorderColor="#000080"
     BorderWidth="2px"
     AutoGenerateColumns="False"
     Width="90%"
     HorizontalAlign="Center"
     AllowPaging="True"
     PageSize="5"
     PagerStyle-Visible="False">
  <HeaderStyle HorizontalAlign="Center"
    CssClass="tableHeader" />
  <ItemStyle CssClass="tableCellNormal" />
  <AlternatingItemStyle CssClass="tableCellAlternating" />
  <Columns>
   <asp:BoundColumn HeaderText="Title"
    DataField="Title" />
   <asp:BoundColumn HeaderText="ISBN"
    DataField="ISBN"
    ItemStyle-HorizontalAlign="Center" />
   <asp:BoundColumn HeaderText="Publisher"
    DataField="Publisher"
    ItemStyle-HorizontalAlign="Center" />
  </Columns>
 </asp:DataGrid>
 <br />
 <table width="40%" border="0" align="center">
  <tr>
   <td align="center">
    <input id="btnFirst" runat="server" type="button"
    value="First"
    onserverclick="btnFirst_ServerClick">
   </td>
   <td align="center">
    <input id="btnPrev" runat="server" type="button"
    value="Prev"
    onserverclick="btnPrev_ServerClick">
   </td>
   <td align="center">
    <input id="btnNext" runat="server" type="button"
    value="Next"
    onserverclick="btnNext_ServerClick">
   </td>
   <td align="center">
    <input id="Last" runat="server" type="button"
    value="Last"
    onserverclick="btnLast_ServerClick">
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 2-21. DataGrid with first/last and next/previous navigation code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02DatagridWithFirstLastNavVB.aspx
 ''' </summary>
 Partial Class CH02DatagridWithFirstLastNavVB
  Inherits System.Web.UI.Page
```

```
'''*****************************************************************************
''' <summary>
''' This routine provides the event handler for the page load event. It
''' is responsible for initializing the controls on the page.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub Page_Load(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles Me.Load
 If (Not Page.IsPostBack) Then
  bindData()
 End If
End Sub 'Page_Load
'''*****************************************************************************
''' <summary>
''' This routine provides the event handler for the first button click
''' event. It is responsible for setting the page index to the first
''' page and rebinding the data.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnFirst_ServerClick(ByVal sender As Object, _
      ByVal e As System.EventArgs)
 'set new page index and rebind the data
 If (dgBooks.CurrentPageIndex > 0) Then
  dgBooks.CurrentPageIndex = 0
  bindData()
 End If
End Sub 'btnFirst_ServerClick

'''*****************************************************************************
''' <summary>
''' This routine provides the event handler for the previous button click
''' event. It is responsible for setting the page index to the previous
''' page and rebinding the data.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnPrev_ServerClick(ByVal sender As Object, _
      ByVal e As System.EventArgs)
 'set new page index and rebind the data
 If (dgBooks.CurrentPageIndex > 0) Then
  dgBooks.CurrentPageIndex -= 1
  bindData()
 End If
End Sub 'btnPrev_ServerClick

'''*****************************************************************************
''' <summary>
''' This routine provides the event handler for the next button click
''' event. It is responsible for setting the page index to the next
''' page and rebinding the data.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnNext_ServerClick(ByVal sender As Object, _
      ByVal e As System.EventArgs)
 'set new page index and rebind the data
 If (dgBooks.CurrentPageIndex < dgBooks.PageCount - 1) Then
  dgBooks.CurrentPageIndex += 1
  bindData()
 End If
End Sub 'btnNext_ServerClick
'''**************************************************************************
''' <summary>
''' This routine provides the event handler for the last button click
''' event. It is responsible for setting the page index to the last
''' page and rebinding the data.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnLast_ServerClick(ByVal sender As Object, _
      ByVal e As System.EventArgs)
 'set new page index and rebind the data
 If (dgBooks.CurrentPageIndex < dgBooks.PageCount - 1) Then
  dgBooks.CurrentPageIndex = dgBooks.PageCount - 1
  bindData()
 End If
End Sub 'btnLast_ServerClick

'''**************************************************************************
''' <summary>
''' This routine queries the database for the data to displayed and binds
''' it to the datagrid
''' </summary>
Private Sub bindData()
 Dim dbConn As OleDbConnection = Nothing
 Dim da As OleDbDataAdapter = Nothing
 Dim dSet As DataSet = Nothing
 Dim strConnection As String
 Dim strSQL As String

 Try
  'get the connection string from web.config and open a connection
  'to the database
  strConnection = ConfigurationManager. _
  ConnectionStrings("dbConnectionString").ConnectionString
  dbConn = New OleDbConnection(strConnection)
  dbConn.Open()
```

```
    'build the query string and get the data from the database
    strSQL = "SELECT Title, ISBN, Publisher " & _
     "FROM Book " & _
     "ORDER BY Title"
    da = New OleDbDataAdapter(strSQL, dbConn)
    dSet = New DataSet
    da.Fill(dSet)

    'set the source of the data for the datagrid control and bind it
    dgBooks.DataSource = dSet
    dgBooks.DataBind()
   Finally
    'cleanup
    If (Not IsNothing(dbConn)) Then
    dbConn.Close()
    End If
   End Try
  End Sub 'bindData
 End  Class  'CH02DatagridWithFirstLastNavVB
End Namespace
```

Example 2-22. DataGrid with first/last and next/previous navigation code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02DatagridWithFirstLastNavCS.aspx
 /// </summary>
 public  partial  class  CH02DatagridWithFirstLastNavCS  :  System.Web.UI.Page
 {
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   if (!Page.IsPostBack)
   {
    bindData();
   }
  } // Page_Load

  ///***************************************************************************
  /// <summary>
  /// This routine provides the event handler for the first button click
  /// event. It is responsible for setting the page index to the first
  /// page and rebinding the data.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnFirst_ServerClick(Object sender,
        System.EventArgs e)
  {
   // set new page index and rebind the data
   if (dgBooks.CurrentPageIndex > 0)
   {
    dgBooks.CurrentPageIndex = 0;
    bindData();
   }
  } // btnFirst_ServerClick
  ///***************************************************************************
  /// <summary>
  /// This routine provides the event handler for the previous button click
  /// event. It is responsible for setting the page index to the previous
  /// page and rebinding the data.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnPrev_ServerClick(Object sender,
        System.EventArgs e)
  {
   // set new page index and rebind the data
   if (dgBooks.CurrentPageIndex > 0)
   {
    dgBooks.CurrentPageIndex -= 1;
    bindData();
   }
  } // btnPrev_ServerClick
  ///***************************************************************************
  /// <summary>
  /// This routine provides the event handler for the next button click
  /// event. It is responsible for setting the page index to the next
  /// page and rebinding the data.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
```

```
protected void btnNext_ServerClick(Object sender,
      System.EventArgs e)
{
 // set new page index and rebind the data
 if (dgBooks.CurrentPageIndex < dgBooks.PageCount - 1)
 {
  dgBooks.CurrentPageIndex += 1;
  bindData();
 }
} // btnNext_ServerClick

///**************************************************************************
/// <summary>
/// This routine provides the event handler for the last button click
/// event. It is responsible for setting the page index to the last
/// page and rebinding the data.
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void btnLast_ServerClick(Object sender,
      System.EventArgs e)
{
 // set new page index and rebind the data
 if (dgBooks.CurrentPageIndex < dgBooks.PageCount - 1)
 {
  dgBooks.CurrentPageIndex = dgBooks.PageCount - 1;
  bindData();
 }
} // btnLast_ServerClick
///**************************************************************************
/// <summary>
/// This routine queries the database for the data to displayed and binds
/// it to the datagrid
/// </summary>
private void bindData()
{
 OleDbConnection dbConn = null;
 OleDbDataAdapter da = null;
 DataSet dSet = null;
 String strConnection;
 String strSQL;

 try
 {
  // get the connection string from web.config and open a connection
  // to the database
  strConnection = ConfigurationManager.
  ConnectionStrings["dbConnectionString"].ConnectionString;
  dbConn = new OleDbConnection(strConnection);
  dbConn.Open();
```

```
     // build the query string and get the data from the database
     strSQL = "SELECT Title, ISBN, Publisher " +
       "FROM Book " +
       "ORDER BY Title";
     da = new OleDbDataAdapter(strSQL, dbConn);
     dSet = new DataSet();
     da.Fill(dSet);

     // set the source of the data for the datagrid control and bind it
     dgBooks.DataSource = dSet;
     dgBooks.DataBind();
    }
    finally
    {
     // cleanup
     if (dbConn != null)
     {
     dbConn.Close();
     }
    }
   } // bindData
  }  //  CH02DatagridWithFirstLastNavCS
 }
```

PREV

Recipe 2.10. Adding Direct Page Navigation to a DataGrid

Problem

You need to display data in a table, but the database that stores it has more rows than can fit on a single page, and you want to allow the user to select the page to display.

Solution

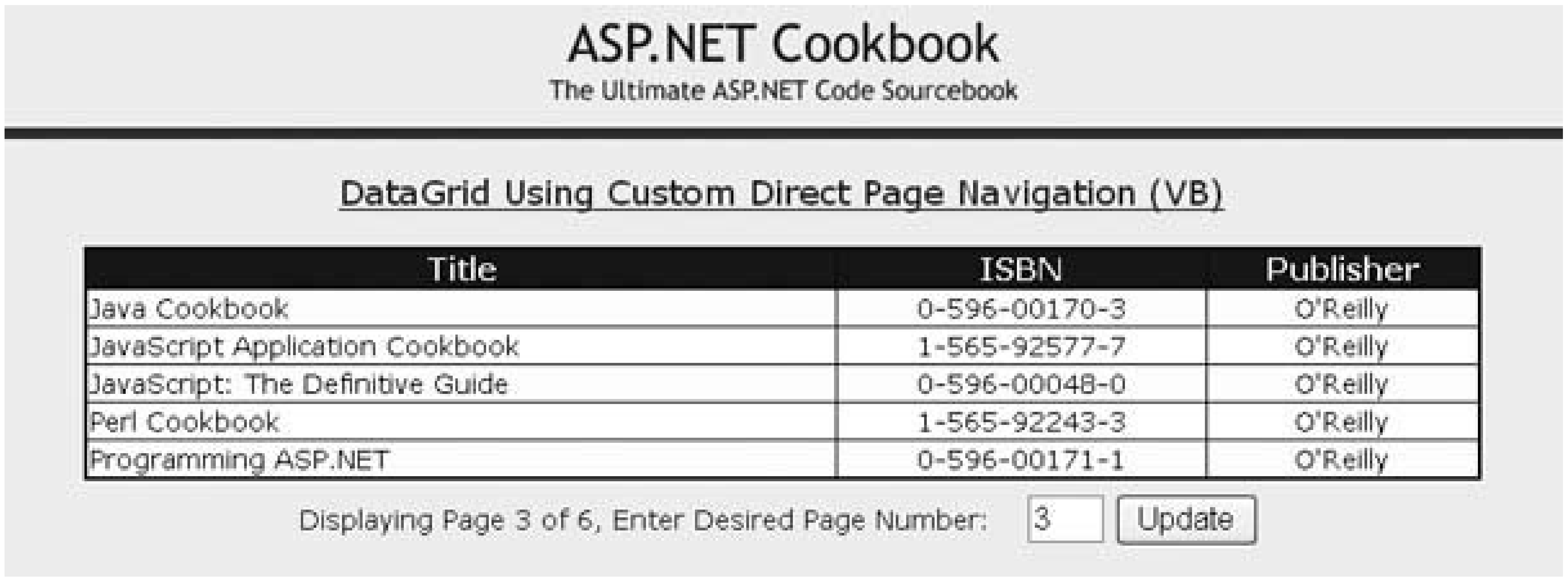
The simplest solution is to use a `DataGrid` control and its `PagerStyle-Mode` and `PagerStyle-PageButton` attributes to enable page selection. This approach produces output like that shown in Figure 2-9. To create an application that employs this approach, start by implementing Recipe 2.7 with the changes to the `DataGrid` tag shown here:

```
PagerStyle-Mode="NumericPages"
PagerStyle-PageButtonCount="5"
```

Figure 2-9. Output of simple solution to DataGrid direct paging

A more flexible solution is to hide the pager provided by the `DataGrid` control and implement your own user interface for the pagination controls. This approach allows the user, for example, to input a page number in a text box and then click a button to display the data as shown in Figure 2-10. Examples 2-23, 2-24 through 2-25 show the *.aspx* and code-behind files for an application that produces this result.

Figure 2-10. Custom direct page navigation with a DataGrid output



ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

DataGrid Using Custom Direct Page Navigation (VB)

Title	ISBN	Publisher
Java Cookbook	0-596-00170-3	O'Reilly
JavaScript Application Cookbook	1-565-92577-7	O'Reilly
JavaScript: The Definitive Guide	0-596-00048-0	O'Reilly
Perl Cookbook	1-565-92243-3	O'Reilly
Programming ASP.NET	0-596-00171-1	O'Reilly

Displaying Page 3 of 6, Enter Desired Page Number: 3 Update

Discussion

In the simple solution, setting the `PagerStyle-Mode` attribute to `NumericPages` causes the `DataGrid` to be rendered with page number buttons for navigating through the grid. The `PagerStyle-ButtonCount` attribute defines the number of "page buttons" that are output. If more pages are available than can be displayed, an ellipsis (...) is displayed at the end(s) of the page buttons containing additional pages. As shown in Figure 2-9, additional pages of data are available beyond Section 1.3. Clicking on the ellipsis will update the data in the `DataGrid` and display the next block of available pages for navigation. In our example, clicking on the ellipsis will cause Section 1.3.3 to be displayed in the `DataGrid` and Section 1.3.3, Section 1.3.4, Section 1.4, Section 1.4.2Section 1.4.4 will be available for direct navigation.

The more flexible solution enables paging in the grid but, with the code shown next, hides the pager provided by the `DataGrid`. Enabling pagination is required to have the `DataGrid` provide the infrastructure needed to perform the paging. Hiding the pager provides you with the ability to implement your own user interface for the pagination controls.

```
AllowPaging="True"
PageSize="5"
PagerStyle-Visible="False">
```

The pagination controls provided in our example consist of a label to display the current page information, a text box to allow the user to enter the desired page number, and a button to initiate the page change. These controls are placed below the `DataGrid` in a row of the table containing the grid.

Like many of the previous examples in this chapter, the code-behind's bindData method queries the database to fill a DataSet. Additionally, with the following code, it updates the label used to display the "page x of y" information and to prompt the user to enter a page number.

VB

```
 lblPager.Text = "Displaying Page " & _
    CStr(dgBooks.CurrentPageIndex + 1) & " of " & _
    CStr(dgBooks.PageCount) & _
    ", Enter Desired Page Number:"
```

C#

```
 lblPager.Text = "Displaying Page " +
    Convert.ToString(dgBooks.CurrentPageIndex + 1) +
    " of " + Convert.ToString(dgBooks.PageCount) +
    ", Enter Desired Page Number:";
```

The btnDisplayPage_Click method in the code-behind provides the server-side event handler for the button click event. This method retrieves from the text box the page number entered by the user anc decrements it by 1, sets the CurrentPageIndex for the DataGrid, and rebinds the data. The page number must be decremented by 1 because the DataGrid pages are zero-based.

The pagination controls we provide here are simple. Almost any user interface can be constructed using HTML and tied into the DataGrid pagination functionality.

Production code should include validation of the page number entered by the user. This can be implemented using a RangeValidator control as described in Recipe 3.2.

See Also

Recipes 2.7 and 2.8 for implementing first/last and next/previous page navigation

Example 2-23. Custom direct page navigation with a DataGrid (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH02DatagridWithDirectPageNavVB2.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH02DatagridWithDirectPageNavVB2"
  Title="DataGrid With Direct Page Navigation 2" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  DataGrid Using Custom Direct Page Navigation (VB)
 </div>
 <asp:DataGrid ID="dgBooks" Runat="server"
     BorderColor="#000080" BorderWidth="2px"
     AutoGenerateColumns="False"
     Width="90%"
     HorizontalAlign="center"
     AllowPaging="True"
     PageSize="5"
     PagerStyle-Visible="False">
  <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
  <ItemStyle CssClass="tableCellNormal" />
  <AlternatingItemStyle CssClass="tableCellAlternating" />
  <Columns>
   <asp:BoundColumn HeaderText="Title"
     DataField="Title" />
   <asp:BoundColumn HeaderText="ISBN"
     DataField="ISBN"
     ItemStyle-HorizontalAlign="Center" />
   <asp:BoundColumn HeaderText="Publisher"
     DataField="Publisher"
     ItemStyle-HorizontalAlign="Center" />
  </Columns>
 </asp:DataGrid>
 <br />
 <table width="70%" border="0" align="center">
  <tr>
   <td>
    <asp:Label ID="lblPager" Runat="server" CssClass="pagerText" />
   </td>
   <td>
    <asp:TextBox ID="txtNewPageNumber" Runat="server" Width="40" />
   </td>
   <td>
    <asp:Button ID="btnDisplayPage" Runat="server"
       Text="Update" OnClick="btnDisplayPage_Click" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 2-24. Custom direct page navigation with a DataGrid code-behin (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02DatagridWithDirectPageNavVB2.aspx
 ''' </summary>
 Partial  Class  CH02DatagridWithDirectPageNavVB2
  Inherits System.Web.UI.Page

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub  Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   If (Not Page.IsPostBack) Then
    bindData()
   End If
  End Sub 'Page_Load

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the display page button
  ''' click event of the datagrid. It is responsible for setting the page
  ''' index to the entered page and rebinding the data.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnDisplayPage_Click(ByVal sender As Object, _
          ByVal e As System.EventArgs) _
   'set new page index and rebind the data
   'NOTE: The page numbers used by the datagrid control are 0 based
   '      so adjust the user enter page number to be 0 based
   dgBooks.CurrentPageIndex = CInt(txtNewPageNumber.Text) - 1
   bindData()
  End Sub 'btnDisplayPage_Click
```

```
   '''***************************************************************************
  ''' <summary>
  ''' This routine queries the database for the data to displayed and binds
  ''' it to the datagrid
  ''' </summary>
  Private Sub bindData()

   Dim dbConn As OleDbConnection = Nothing
   Dim da As OleDbDataAdapter = Nothing
   Dim dSet As DataSet = Nothing
   Dim strConnection As String
   Dim strSQL As String

   Try
    'get the connection string from web.config and open a connection
    'to the database
    strConnection = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDb.OleDbConnection(strConnection)
    dbConn.Open()

    'build the query string and get the data from the database
    strSQL = "SELECT Title, ISBN, Publisher " & _
     "FROM Book " & _
     "ORDER BY Title"
```

```
da = New OleDbDataAdapter(strSQL, dbConn)
 dSet = New DataSet
 da.Fill(dSet)

 'set the source of the data for the datagrid control and bind it
 dgBooks.DataSource = dSet
 dgBooks.DataBind()

 'update label on custom pager to show current page and total pages
 lblPager.Text = "Displaying Page " & _
    CStr(dgBooks.CurrentPageIndex + 1) & " of " & _
    CStr(dgBooks.PageCount) & _
    ", Enter Desired Page Number:"

 Finally
  'cleanup
  If (Not IsNothing(dbConn)) Then
   dbConn.Close()
  End If
  End Try
   End Sub 'bindData
 End  Class  'CH02DatagridWithDirectPageNavVB2
End Namespace
```

Example 2-25. Custom direct page navigation with a DataGrid code-behin (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
  ///*****************************************************************
  /// <summary>
  /// This class provides the code behind for
  ///  CH02DatagridWithDirectPageNavCS2.aspx
  /// </summary>
  public partial class CH02DatagridWithDirectPageNavCS2 : System.Web.UI.Page
  {
  ///**************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   if (!Page.IsPostBack)
   {
    bindData();
   }
  } //  Page_Load

  ///**************************************************************
  /// <summary>
  /// This routine provides the event handler for the display page button
  /// click event of the datagrid. It is responsible for setting the page
  /// index to the entered page and rebinding the data.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnDisplayPage_Click(Object sender,
             System.EventArgs e)
  {
   // set new page index and rebind the data
   // NOTE: The page numbers used by the datagrid control are 0 based
   // so adjust the user enter page number to be 0 based
   dgBooks.CurrentPageIndex = Convert.ToInt32(txtNewPageNumber.Text) - 1;
   bindData();
  } // btnDisplayPage_Click

  ///*****************************************************************
  /// <summary>
  /// This routine queries the database for the data to displayed and binds
  /// it to the datagrid
  /// </summary>
  private void bindData()
  {
   OleDbConnection dbConn = null;
   OleDbDataAdapter da = null;
   DataSet dSet = null;
   String strConnection;
   String strSQL;

   try
   {
    // get the connection string from web.config and open a connection
    // to the database
    strConnection = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(strConnection);
    dbConn.Open();

    // build the query string and get the data from the database
    strSQL = "SELECT Title, ISBN, Publisher " +
     "FROM Book " +
     "ORDER BY Title";

    da = new OleDbDataAdapter(strSQL, dbConn);
    dSet = new DataSet();
    da.Fill(dSet);
    // set the source of the data for the datagrid control and bind it
    dgBooks.DataSource = dSet;
    dgBooks.DataBind();

    //update label on custom pager to show current page and total pages
    lblPager.Text = "Displaying Page " +
      Convert.ToString(dgBooks.CurrentPageIndex + 1) +
      " of " + Convert.ToString(dgBooks.PageCount) +
      ", Enter Desired Page Number:";
   }

   finally
   {
    // cleanup
    if (dbConn != null)
    {
    dbConn.Close();
```

```
        }
      }
    } // bindData
    }  //  CH02DatagridWithDirectPageNavCS2
}
```

PREV

Recipe 2.11. Sorting Data in Ascending/Descending Order Within a DataGrid

Problem

You are displaying a table of data in a `DataGrid` , and you want to let the user sort the data and change the sort order by clicking on the column headers.

Solution

Enable the `DataGrid` control's sorting features, and add custom coding to support the sorting along with an indication of the current sort column and order.

In the *.aspx* file, enable the `DataGrid` control's sorting features.

In the code-behind class for the page, use the .NET language of your choice to:

1. Create a data-binding method (`bindData` in our example) that does the following:

    a. Generates the SQL statement required to get the data from the database with an `ORDER BY` clause based on the value of the `sortExpression` parameter

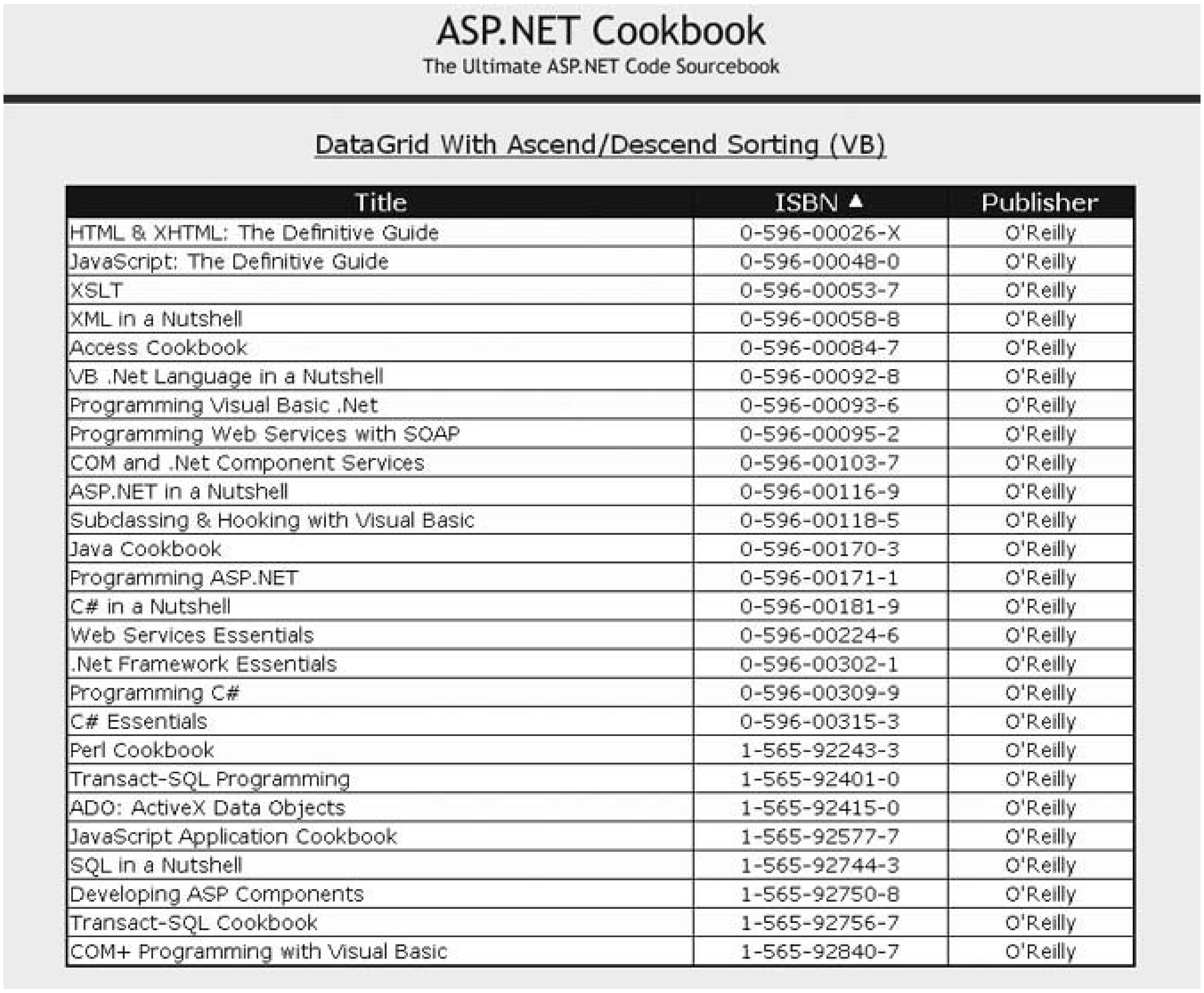
    b. Fills a `DataTable` with the ordered data from the database

    c. Outputs an image indicating the sort column and sort order beside the current sort column heading

2. Call the data-binding method from the `Page_Load` method (to support the initial display of the grid) and from the event that is fired when the user clicks on a column header (the `dgBooks_SortCommand` event in our example).

Figure 2-11 shows the appearance of a typical `DataGrid` sorted by title in ascending order, the information in the first column. Examples 2-26 , 2-27 through 2-28 show the *.aspx* and code-behind files for an example application that produces this result.

Figure 2-11. DataGrid with ascending/descending sorting output



ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

DataGrid With Ascend/Descend Sorting (VB)

Title	ISBN ▲	Publisher
HTML & XHTML: The Definitive Guide	0-596-00026-X	O'Reilly
JavaScript: The Definitive Guide	0-596-00048-0	O'Reilly
XSLT	0-596-00053-7	O'Reilly
XML in a Nutshell	0-596-00058-8	O'Reilly
Access Cookbook	0-596-00084-7	O'Reilly
VB .Net Language in a Nutshell	0-596-00092-8	O'Reilly
Programming Visual Basic .Net	0-596-00093-6	O'Reilly
Programming Web Services with SOAP	0-596-00095-2	O'Reilly
COM and .Net Component Services	0-596-00103-7	O'Reilly
ASP.NET in a Nutshell	0-596-00116-9	O'Reilly
Subclassing & Hooking with Visual Basic	0-596-00118-5	O'Reilly
Java Cookbook	0-596-00170-3	O'Reilly
Programming ASP.NET	0-596-00171-1	O'Reilly
C# in a Nutshell	0-596-00181-9	O'Reilly
Web Services Essentials	0-596-00224-6	O'Reilly
.Net Framework Essentials	0-596-00302-1	O'Reilly
Programming C#	0-596-00309-9	O'Reilly
C# Essentials	0-596-00315-3	O'Reilly
Perl Cookbook	1-565-92243-3	O'Reilly
Transact-SQL Programming	1-565-92401-0	O'Reilly
ADO: ActiveX Data Objects	1-565-92415-0	O'Reilly
JavaScript Application Cookbook	1-565-92577-7	O'Reilly
SQL in a Nutshell	1-565-92744-3	O'Reilly
Developing ASP Components	1-565-92750-8	O'Reilly
Transact-SQL Cookbook	1-565-92756-7	O'Reilly
COM+ Programming with Visual Basic	1-565-92840-7	O'Reilly

Discussion

The `DataGrid` control provides the basic plumbing required to support sorting. It will generate the links for the column headers to raise the `SortCommand` server-side event when a column header is clicked. The `DataGrid` does not provide the code required to perform the actual sorting, but very little code is required to complete that job.

To enable sorting, the `AllowSorting` attribute of the `DataGrid` element must be set to `true` . In addition, the `SortExpression` attribute of the `BoundColumn` element must be set to the expression that will be used in your code to perform the sorting. This would normally be set to the name of the database column displayed in the `DataGrid` column; however, it can be set to any value required by your code to perform the sorting.

Your code will need to perform the actual sorting. For example, the application that we developed for this recipe supports sorting in a centralized manner by using `sortExpression` and `sortOrder` parameters with its `bindData` method. The parameters are used in the `ORDER BY` clause of the SQL statement.

Your code will need to perform the sorting in ascending and descendingorder as well as visually indicating the sort order, which requires a bit of coding. This is driven by needing to know the curren sort column and the sort order to determine what needs to be done when the user clicks a column header. From here on, it's useful to examine our example application to see how we've juggled these needs.

We've added an enumeration and several constants to the top of the code-behind class shown in Examples 2-27 (VB) and 2-28 (C#). The `enuSortOrder` enumeration defines the available sort orders used to store and compare sort order data. The `sortExpression` and `columnTitle` arrays are used to define the sort expression and column title for each column in the`DataGrid` . As their names imply, the `VS_CURRENT_SORT_EXPRESSION` and `VS_CURRENT_SORT_ORDER` constants define the names of variables stored in the `ViewState` to track the current sort expression and order between page submittals.

In our example application, the `Page_Load` method performs two operations. First, the view state variables used to store the current sort expression and sort order (`VS_CURRENT_SORT_EXPRESSION` and `VS_CURRENT_SORT_ORDER`) are set to their default values. For this example, the title column is sorted ir ascending order by default. Second, the `bindData` method is called to pass the current sort expression and sort order.

Two features of the `bindData` method are worth special note. First, the SQL statement used to query the database includes an `ORDER BY` clause that reflects the current sort order.

Second, the `bindData` method loops through each column in the grid, determines which column is the sort column, and marks the sort order for the column. The sort column is determined by comparing the current sort expression passed to the `bindData` method with the sort expression for each of the columns. The sort expression for the one column that matches the current sort expression is the sort column. After finding the sort column, the sort order is checked to determine whether the ascending or descending image should be output in the header for the sort column. Finally, the header text is set to the title for the column, and when relevant, the HTML image tag used to indicate the sort orde is set. For columns that are not the current sort column, the image-related HTML is set to an empty string.

When the user clicks a column header in the grid, the`dgBooks_SortCommand` method is called and that method determines the changes that need to be made prior to rebinding the data. The first step is to get the current sort expression and sort order from the view state, as shown here:

```
currentSortExpression = CStr(viewstate(VS_CURRENT_SORT_EXPRESSION))
currentSortOrder = CType(viewstate(VS_CURRENT_SORT_ORDER), enuSortOrder)

currentSortExpression = (String)(this.ViewState[VS_CURRENT_SORT_EXPRESSION]);
currentSortOrder = (enuSortOrder)(this.ViewState[VS_CURRENT_SORT_ORDER]);
```

After getting the current information, we determine if a column other than the current sort column has been clicked. If so, the clicked column is set as the new sort column, and the sortorder is set to ascending. If not, we need to change the sort order of the original column.

After determining the sort expression and sort order, the view state is updated to reflect what will be the current information once the page is rendered.

Finally, the data is rebound to the grid by calling the bindData method with the new sort information.

When working with this recipe's sample application, the following code must be placed *before* the DataBind statement. As a general rule, changes of this sort made to a DataGrid control placed after data binding may not be displayed as intended.

VB

```
Dim col As DataGridColumn

 …

For index = 0 To dgBooks.Columns.Count - 1
 col = dgBooks.Columns(index)

 'check to see if this is the sort column
 If (col.SortExpression = sortExpression) Then
  'this is the sort column so determine whether the ascending or
  'descending image needs to be included
  If (sortOrder = enuSortOrder.soAscending) Then
    colImage = " <img src='images/sort_ascending.gif' border='0'>"
  Else
    colImage = " <img src='images/sort_descending.gif' border='0'>"
  End if
 Else
  'This is not the sort column so include no image html
  colImage = ""
 End If 'If (col.SortExpression = sortExpression)

 'set the title for the column
 col.HeaderText = columnTitle(index) & colImage
Next index

for (index = 0; index < dgBooks.Columns.Count; index++)
{
 col = dgBooks.Columns[index];
 // check to see if this is the sort column
 if (col.SortExpression == sortExpression)
 {
  // this is the sort column so determine whether the ascending or
  // descending image needs to be included
  if (sortOrder == enuSortOrder.soAscending)
  {
    colImage = " <img src='images/sort_ascending.gif' border='0'>";
  }
  else
  {
   colImage = " <img src='images/sort_descending.gif border='0'>";
  }
 }
 else
```

```
          {
           // This is not the sort column so include no image html
           colImage = "";
          } // if (col.SortExpression == sortExpression)

          // set the title for the column
          col.HeaderText = columnTitle[index] + colImage;
         } // for index
```

Example 2-26. DataGrid with ascending/descending sorting (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH02DatagridAscDescSortingVB.aspx.vb"
   Inherits="ASPNetCookbook.VBExamples.CH02DatagridAscDescSortingVB"
  Title="DataGrid With Ascend/Descend Sorting " %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  DataGrid With Ascend/Descend Sorting (VB)
 </div>
 <asp:DataGrid id="dgBooks" runat="server"
     BorderColor="#000080"
     BorderWidth="2px"
     HorizontalAlign="Center"
     AutoGenerateColumns="False"
     Width="90%"
     AllowSorting="True"
     OnSortCommand="dgBooks_SortCommand" >
    <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
    <ItemStyle cssClass="tableCellNormal" />
    <AlternatingItemStyle cssClass="tableCellAlternating" />

    <Columns>
    <asp:BoundColumn DataField="Title"
     SortExpression="Title" />
    <asp:BoundColumn DataField="ISBN"
     ItemStyle-HorizontalAlign="Center"
     SortExpression="ISBN" />
    <asp:BoundColumn DataField="Publisher"
     ItemStyle-HorizontalAlign="Center"
     SortExpression="Publisher" />
   </Columns>
 </asp:DataGrid>
</asp:Content>
```

Example 2-27. DataGrid with ascending/descending sorting code-behind (.vb)

```
Option Explicit
On Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02DatagridAscDescSortingVB.aspx
 ''' </summary>
 Partial Class CH02DatagridAscDescSortingVB
  Inherits System.Web.UI.Page

   'the following enumeration is used to define the sort orders
   Private Enum enuSortOrder
  soAscending = 0
  soDescending = 1
   End Enum

   'strings to use for the sort expressions and column title
   'separate arrays are used to support the sort expression and titles
   'being different
   Private ReadOnly sortExpression() As String = {"Title", "ISBN", "Publisher"}
   Private ReadOnly columnTitle() As String = {"Title", "ISBN", "Publisher"}

   'the names of the variables placed in the viewstate
   Private Const VS_CURRENT_SORT_EXPRESSION As String = "currentSortExpression"
   Private Const VS_CURRENT_SORT_ORDER As String = "currentSortOrder"

   '''*****************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the page load event. It
   ''' is responsible for initializing the controls on the page.
   ''' </summary>
   '''
   ''' <param name="sender">Set to the sender of the event</param>
   ''' <param name="e">Set to the event arguments</param>
   Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
  Dim defaultSortExpression As String
  Dim defaultSortOrder As enuSortOrder

  If (Not Page.IsPostBack) Then
   'sort by title, ascending as the default
```

```
    defaultSortExpression = sortExpression(0)
    defaultSortOrder = enuSortOrder.soAscending

    'store current sort expression and order in the viewstate then
    'bind data to the DataGrid
    ViewState(VS_CURRENT_SORT_EXPRESSION) = defaultSortExpression
    ViewState(VS_CURRENT_SORT_ORDER) = defaultSortOrder
    bindData(defaultSortExpression, _
      defaultSortOrder)
  End If
End Sub 'Page_Load

'''*****************************************************************
''' <summary>
''' This routine provides the event handler for the datagrid sort event.
''' It is responsible re-binding the data to the datagrid by the selected
''' column.
''' </summary>
'''
''' <param name="source">Set to the source of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub dgBooks_SortCommand(ByVal source As Object, _
         ByVal e As DataGridSortCommandEventArgs)

    Dim newSortExpression As String
    Dim currentSortExpression As String
    Dim currentSortOrder As enuSortOrder

    'get the current sort expression and order from the viewstate
    currentSortExpression = CStr(ViewState(VS_CURRENT_SORT_EXPRESSION))
    currentSortOrder = CType(ViewState(VS_CURRENT_SORT_ORDER), enuSortOrder)

    'check to see if this is a new column or the sort order
    'of the current column needs to be changed.
    newSortExpression = e.SortExpression
    If (newSortExpression = currentSortExpression) Then
  'sort column is the same so change the sort order
  If (currentSortOrder = enuSortOrder.soAscending) Then
    currentSortOrder = enuSortOrder.soDescending
  Else
    currentSortOrder = enuSortOrder.soAscending
  End If
    Else
  'sort column is different so set the new column with ascending
  'sort order
  currentSortExpression = newSortExpression
  currentSortOrder = enuSortOrder.soAscending
  End If

  'update the view state with the new sort information
  ViewState(VS_CURRENT_SORT_EXPRESSION) = currentSortExpression
  ViewState(VS_CURRENT_SORT_ORDER) = currentSortOrder
```

```
'rebind the data in the datagrid
bindData(currentSortExpression, _
   currentSortOrder)
 End Sub 'dgBooks_SortCommand

 '''*********************************************************************
 ''' <summary>
 ''' This routine queries the database for the data to displayed and binds
 ''' it to the datagrid
 ''' </summary>
 '''
 ''' <param name="sortExpression">Set to the sort expression to use for
 ''' sorting the data</param>
 ''' <param name="sortOrder">Set to the requried sort order</param>
 Private Sub bindData(ByVal sortExpression As String, _
  ByVal sortOrder As enuSortOrder)
Dim dbConn As OleDbConnection = Nothing
Dim da As OleDbDataAdapter = Nothing
Dim dTable As DataTable = Nothing
Dim strConnection As String
Dim strSQL As String
Dim index As Integer
Dim col As DataGridColumn = Nothing
Dim colImage As String
Dim strSortOrder As String

Try
'get the connection string from web.config and open a connection
'to the database
strConnection = ConfigurationManager. _
 ConnectionStrings("dbConnectionString").ConnectionString
dbConn = New OleDbConnection(strConnection)
dbConn.Open()

'build the query string and get the data from the database
If (sortOrder = enuSortOrder.soAscending) Then
 strSortOrder = " ASC"
Else
 strSortOrder = " DESC"
End If

strSQL = "SELECT Title, ISBN, Publisher " & _
   "FROM Book " & _
   "ORDER BY " & sortExpression & _
   strSortOrder

da = New OleDbDataAdapter(strSQL, dbConn)
dTable = New DataTable
da.Fill(dTable)

   'loop through the columns in the datagrid updating the heading to
  'mark which column is the sort column and the sort order
  For index = 0 To dgBooks.Columns.Count - 1
   col = dgBooks.Columns(index)

   'check to see if this is the sort column
   If (col.SortExpression = sortExpression) Then
   'this is the sort column so determine whether the ascending or
   'descending image needs to be included
   If (sortOrder = enuSortOrder.soAscending) Then
    colImage = " <img src='images/sort_ascending.gif' border='0'>"
   Else
    colImage = " <img src='images/sort_descending.gif' border='0'>"
   End If
    Else
   'This is not the sort column so include no image html
   colImage = ""
    End If 'If (col.SortExpression = sortExpression)

    'set the title for the column
    col.HeaderText = columnTitle(index) & colImage
   Next index

   'set the source of the data for the datagrid control and bind it
   dgBooks.DataSource = dTable
         dgBooks.DataBind()

  Finally
   'cleanup
   If (Not IsNothing(dbConn)) Then
    dbConn.Close()
   End If
   End Try
  End Sub 'bindData
  End Class 'CH02DatagridAscDescSortingVB
End Namespace
```

Example 2-28. DataGrid with ascending/descending sorting code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02DatagridAscDescSortingCS.aspx
 /// </summary>
 public partial class CH02DatagridAscDescSortingCS : System.Web.UI.Page
 {
  // the following enumeration is used to define the sort orders
  private enum enuSortOrder
  {
   soAscending = 0,
   soDescending = 1
  }

  // strings to use for the sort expressions and column title
  // separate arrays are used to support the sort expression and titles
  // being different
  static readonly String[] sortExpression =
      new String[] { "Title", "ISBN", "Publisher" };
  static readonly String[] columnTitle =
      new String[] { "Title", "ISBN", "Publisher" };

  // the names of the variables placed in the viewstate
  static readonly String VS_CURRENT_SORT_EXPRESSION = "currentSortExpression";
  static readonly String VS_CURRENT_SORT_ORDER = "currentSortOrder";

  ///*****************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event. It
  /// is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  private void Page_Load(object sender, System.EventArgs e)
  {
   String defaultSortExpression;
   enuSortOrder defaultSortOrder;

   if (!Page.IsPostBack)
   {
    // sort by title, ascending as the default
    defaultSortExpression = sortExpression[0];
    defaultSortOrder = enuSortOrder.soAscending;

    // bind data to the DataGrid
    this.ViewState.Add(VS_CURRENT_SORT_EXPRESSION, defaultSortExpression);
    this.ViewState.Add(VS_CURRENT_SORT_ORDER, defaultSortOrder);
    bindData(defaultSortExpression,
     defaultSortOrder);
   }
  } // Page_Load

  ///*****************************************************************
```

```
/// <summary>
/// This routine provides the event handler for the datagrid sort event.
/// It is responsible re-binding the data to the datagrid by the selected
/// column.
/// </summary>
///
/// <param name="source">Set to the source of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void dgBooks_SortCommand(Object source,
     System.Web.UI.WebControls.DataGridSortCommandEventArgs e)
{
 String newSortExpression = null;
 String currentSortExpression = null;

 enuSortOrder currentSortOrder;

 // get the current sort expression and order from the viewstate
 currentSortExpression =
  (String)(this.ViewState[VS_CURRENT_SORT_EXPRESSION]);
 currentSortOrder =
  (enuSortOrder)(this.ViewState[VS_CURRENT_SORT_ORDER]);

 // check to see if this is a new column or the sort order
 // of the current column needs to be changed.
 newSortExpression = e.SortExpression;
 if (newSortExpression.Equals(currentSortExpression))
 {
  // sort column is the same so change the sort order
  if (currentSortOrder == enuSortOrder.soAscending)
  {
  currentSortOrder = enuSortOrder.soDescending;
  }
  else
  {
  currentSortOrder = enuSortOrder.soAscending;
  }
 }
 else
 {
  // sort column is different so set the new column with ascending
  // sort order
  currentSortExpression = newSortExpression;
  currentSortOrder = enuSortOrder.soAscending;
 }

 // update the view state with the new sort information
 this.ViewState.Add(VS_CURRENT_SORT_EXPRESSION, currentSortExpression);
 this.ViewState.Add(VS_CURRENT_SORT_ORDER, currentSortOrder);

 // rebind the data in the datagrid
 bindData(currentSortExpression,
   currentSortOrder);
```

```
} // dgBooks_SortCommand

///*****************************************************************
/// <summary>
/// This routine queries the database for the data to displayed and binds
/// it to the datagrid
/// </summary>
///
/// <param name="sortExpression">Set to the sort expression to use for
///          sorting the data</param>
/// <param name="sortOrder">Set to the requried sort order</param>
private void bindData(String sortExpression,
    enuSortOrder sortOrder)
{
   OleDbConnection dbConn = null;
   OleDbDataAdapter da = null;
   DataTable dTable = null;
   String strConnection = null;
   String strSQL = null;
   int index = 0;
   DataGridColumn col = null;
   String colImage = null;
   String strSortOrder = null;

   try
   {
        // get the connection string from web.config and open a connection
  // to the database
  strConnection = ConfigurationManager.
  ConnectionStrings["dbConnectionString"].ConnectionString;
  dbConn = new OleDbConnection(strConnection);
  dbConn.Open();

  // build the query string and get the data from the database
  if (sortOrder == enuSortOrder.soAscending)
  {
  strSortOrder = " ASC";
  }
  else
  {
  strSortOrder = " DESC";
  }

  strSQL = "SELECT Title, ISBN, Publisher " +
   "FROM Book " +
   "ORDER BY " + sortExpression +
   strSortOrder;

  da = new OleDbDataAdapter(strSQL, dbConn);
  dTable = new DataTable();
  da.Fill(dTable);
```

```
    // loop through the columns in the datagrid updating the heading to
    // mark which column is the sort column and the sort order
    for (index = 0; index < dgBooks.Columns.Count; index++)
    {
    col = dgBooks.Columns[index];
    // check to see if this is the sort column
    if (col.SortExpression == sortExpression)
    {
    // this is the sort column so determine whether the ascending or
    // descending image needs to be included
    if (sortOrder == enuSortOrder.soAscending)
    {
    colImage = " <img src='images/sort_ascending.gif' border='0'>";
    }
    else
    {
    colImage = " <img src='images/sort_descending.gif' border='0'>";
    }
    }
    else
    {
    // This is not the sort column so include no image html
    colImage = "";
    } // if (col.SortExpression == sortExpression)

    // set the title for the column
    col.HeaderText = columnTitle[index] + colImage;
   } // for index

   // set the source of the data for the datagrid control and bind it
   dgBooks.DataSource = dTable;
   dgBooks.DataBind();
  } // try

 finally
 {
  //clean up
  if (dbConn != null)
  {
   dbConn.Close();
  }
   } // finally
 } // bindData
  } //  CH02DatagridAscDescSortingCS
}
```

PREV

Recipe 2.12. Combining Sorting and Paging in a DataGrid

Problem

You are implementing a `DataGrid` with sorting and pagination, and you are having trouble making the two features work together.

Solution

Enable the sorting features of the `DataGrid` control, and add custom code to support the sorting along with an indication of the current sort column and order (see Recipe 2.10 for details). Next, with pagination enabled, add a small amount of custom code to track the sort column and order so they can be maintained between client round trips and used any time rebinding is required. Figure 2-12 shows a typical `DataGrid` with this solution implemented. Examples 2-29 , 2-30 through 2-31 show the *.aspx* file and code-behind files for an application that produces this output.

Figure 2-12. Combining sorting and paging in a DataGrid output

Discussion

Getting sorting and paging to work at the same time is a notorious problem with a `DataGrid` . The key to making it all work is to track the sort column and order so they can be used any time rebinding is required, whether because of a page change or a sort command. Likewise, it is useful to put the sort column and order data in the view state so they are properly maintained between client round trips.

The `DataGrid` provides the basic plumbing for sorting and paging the data displayed in the grid. The `DataGrid` provides a property `(CurrentPageIndex)` that is always available to indicate which page is to be displayed. Unfortunately, the `DataGrid` provides no information regarding the sort column or order, forcing the programmer to track this information outside of the DataGrid so it will be available when performing pagination operations.

The application we've developed for this recipe should give you a good idea of how to handle sorting and paging simultaneously. It tracks the sort column and order so the proper data can be bound to the DataGrid any time rebinding is required, for example, when the user clicks on a row header to resort a column or selects a page from the DataGrid control's built-in navigation control. Refer to Recipes 2.9 and 2.10 for more detailed discussions of the various nuances of this recipe.

See Also

Recipes 2.9 and 2.10

Example 2-29. Combining sorting and paging in a DataGrid (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02DatagridWithSortingAndPagingVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02DatagridWithSortingAndPagingVB"
 Title="DataGrid With Sorting And Paging" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  DataGrid With Sorting And Paging (VB)
 </div>
 <asp:DataGrid
  id="dgBooks"
  runat="server"
  BorderColor="#000080"
  BorderWidth="2px"
  AutoGenerateColumns="False"
  width="90%"
  HorizontalAlign="Center"
  AllowSorting="True"
  AllowPaging="True"
  PageSize="5"
  PagerStyle-Mode="NumericPages"
  PagerStyle-PageButtonCount="5"
  PagerStyle-Position="Bottom"
  PagerStyle-HorizontalAlign="Center"
  PagerStyle-NextPageText="Next"
  PagerStyle-PrevPageText="Prev"
  PagerStyle-CssClass="pagerText"
  OnPageIndexChanged="dgBooks_PageIndexChanged"
  OnSortCommand="dgBooks_SortCommand" >

  <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
  <ItemStyle cssClass="tableCellNormal" />
  <AlternatingItemStyle cssClass="tableCellAlternating" />
  <Columns>
```

```
   <asp:BoundColumn DataField="Title"
     SortExpression="Title" />
   <asp:BoundColumn DataField="ISBN"
     ItemStyle-HorizontalAlign="Center"
     SortExpression="ISBN" />
   <asp:BoundColumn DataField="Publisher"
     ItemStyle-HorizontalAlign="Center"
     SortExpression="Publisher" />

  </Columns>
 </asp:DataGrid>
</asp:Content>
```

Example 2-30. Combining sorting and paging in a DataGrid code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' CH02DatagridWithSortingAndPagingVB.aspx
 ''' </summary>
 Partial Class CH02DatagridWithSortingAndPagingVB
  Inherits System.Web.UI.Page

  'the following enumeration is used to define the sort orders
  Private Enum enuSortOrder
   soAscending = 0
   soDescending = 1
  End Enum

  'strings to use for the sort expressions and column title
  'separate arrays are used to support the sort expression and titles
  'being different
  Private ReadOnly sortExpression() As String = {"Title", "ISBN", "Publisher"}
  Private ReadOnly columnTitle() As String = {"Title", "ISBN", "Publisher"}

  'the names of the variables placed in the viewstate
  Private Const VS_CURRENT_SORT_EXPRESSION As String = "currentSortExpression"
  Private Const VS_CURRENT_SORT_ORDER As String = "currentSortOrder"
```

```
'''*****************************************************************
''' <summary>
''' This routine provides the event handler for the page load event. It
''' is responsible for initializing the controls on the page.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
 Dim defaultSortExpression As String
 Dim defaultSortOrder As enuSortOrder

 If (Not Page.IsPostBack) Then
  'sort by title, ascending as the default
   defaultSortExpression = sortExpression(0)
  defaultSortOrder = enuSortOrder.soAscending

  'store current sort expression and order in the viewstate then
  'bind data to the DataGrid
  ViewState(VS_CURRENT_SORT_EXPRESSION) = defaultSortExpression
  ViewState(VS_CURRENT_SORT_ORDER) = defaultSortOrder
  bindData(defaultSortExpression, _
   defaultSortOrder)
 End If
End Sub 'Page_Load

'''*****************************************************************
''' <summary>
''' This routine provides the event handler for the datagrid sort event.
''' It is responsible re-binding the data to the datagrid by the selected
''' column.
''' </summary>
'''
''' <param name="source">Set to the source of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub dgBooks_SortCommand(ByVal source As Object, _
       ByVal e As DataGridSortCommandEventArgs)
         Dim newSortExpression As String
   Dim currentSortExpression As String
   Dim currentSortOrder As enuSortOrder

   'get the current sort expression and order from the viewstate
   currentSortExpression = CStr(ViewState(VS_CURRENT_SORT_EXPRESSION))
   currentSortOrder = CType(ViewState(VS_CURRENT_SORT_ORDER), enuSortOrder)

   'check to see if this is a new column or the sort oder
   'of the current column needs to be changed.
   newSortExpression = e.SortExpression
   If (newSortExpression = currentSortExpression) Then
  'sort column is the same so change the sort order
  If (currentSortOrder = enuSortOrder.soAscending) Then
```

```
    currentSortOrder = enuSortOrder.soDescending
   Else
    currentSortOrder = enuSortOrder.soAscending
   End If
     Else
    'sort column is different so set the new column with ascending
    'sort order
    currentSortExpression = newSortExpression
    currentSortOrder = enuSortOrder.soAscending
     End If

     'update the view state with the new sort information
     ViewState(VS_CURRENT_SORT_EXPRESSION) = currentSortExpression
     ViewState(VS_CURRENT_SORT_ORDER) = currentSortOrder

     'rebind the data in the datagrid
     bindData(currentSortExpression, _
    currentSortOrder)
    End Sub 'dgBooks_SortCommand

   '''*****************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the page index changed
   ''' event of the datagrid. It is responsible for setting the page index
   ''' from the passed arguments and rebinding the data.
   ''' </summary>
   '''
   ''' <param name="source">Set to the sender of the event</param>
   ''' <param name="e">Set to the event arguments</param>
   Protected Sub dgBooks_PageIndexChanged(ByVal source As Object, _
     ByVal e As System.Web.UI.WebControls.DataGridPageChangedEventArgs)
   Dim currentSortExpression As String
   Dim currentSortOrder As enuSortOrder

   'set new page index and rebind the data
   dgBooks.CurrentPageIndex = e.NewPageIndex

   'get the current sort expression and order from the viewstate
   currentSortExpression = CStr(ViewState(VS_CURRENT_SORT_EXPRESSION))
   currentSortOrder = CType(ViewState(VS_CURRENT_SORT_ORDER), enuSortOrder)
   'rebind the data in the datagrid
   bindData(currentSortExpression, _
      currentSortOrder)
  End Sub 'dgCustomers_PageIndexChanged

  '''*********************************************************************
  ''' <summary>
  ''' This routine queries the database for the data to displayed and binds
  ''' it to the datagrid
  ''' </summary>
  '''
  ''' <param name="sortExpression">Set to the sort expression to use for
```

```
'''      sorting the data</param>
''' <param name="sortOrder">Set to the requried sort order</param>
Private Sub bindData(ByVal sortExpression As String, _
    ByVal sortOrder As enuSortOrder)
 Dim dbConn As OleDbConnection = Nothing
 Dim da As OleDbDataAdapter = Nothing
 Dim dTable As DataTable = Nothing
 Dim strConnection As String
 Dim strSQL As String
 Dim index As Integer
 Dim col As DataGridColumn = Nothing
 Dim colImage As String
 Dim strSortOrder As String

 Try
 'get the connection string from web.config and open a connection
 'to the database
 strConnection = ConfigurationManager. _
  ConnectionStrings("dbConnectionString").ConnectionString
 dbConn = New OleDbConnection(strConnection)
 dbConn.Open()

 'build the query string and get the data from the database
 If (sortOrder = enuSortOrder.soAscending) Then
  strSortOrder = " ASC"
 Else
  strSortOrder = " DESC"
 End If

 strSQL = "SELECT Title, ISBN, Publisher " & _
   "FROM Book " & _
   "ORDER BY " & sortExpression & _
   strSortOrder

          da = New OleDbDataAdapter(strSQL, dbConn)
 dTable = New DataTable
 da.Fill(dTable)

 'loop through the columns in the datagrid updating the heading to
 'mark which column is the sort column and the sort order
 For index = 0 To dgBooks.Columns.Count - 1
  col = dgBooks.Columns(index)

  'check to see if this is the sort column
  If (col.SortExpression = sortExpression) Then
  'this is the sort column so determine whether the ascending or
  'descending image needs to be included
  If (sortOrder = enuSortOrder.soAscending) Then
   colImage = " <img src='images/sort_ascending.gif' border='0'>"
  Else
   colImage = " <img src='images/sort_descending.gif' border='0'>"
  End If
```

```
    Else
    'This is not the sort column so include no image html
    colImage = ""
    End If 'If (col.SortExpression = sortExpression)

    'set the title for the column
    col.HeaderText = columnTitle(index) & colImage
    Next index

    'set the source of the data for the datagrid control and bind it
    dgBooks.DataSource = dTable
    dgBooks.DataBind()

    Finally
   'cleanup
   If (Not IsNothing(dbConn)) Then
    dbConn.Close()
   End If
  End Try
 End Sub 'bindData
  End  Class  'CH02DatagridWithSortingAndPagingVB
End Namespace
```

Example 2-31. Combining sorting and paging in a DataGrid code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
  /// CH02DatagridWithSortingAndPagingCS.aspx
  /// </summary>
   public  partial  class  CH02DatagridWithSortingAndPagingCS  :  System.Web.UI.Page
  {
  // the following enumeration is used to define the sort orders
  private enum enuSortOrder
  {
   soAscending = 0,
   soDescending = 1
  }

  // strings to use for the sort expressions and column title
  // separate arrays are used to support the sort expression and titles
  // being different
  static readonly String[] sortExpression =
         new String[] { "Title", "ISBN", "Publisher" };
  static readonly String[] columnTitle =
            new String[] { "Title", "ISBN", "Publisher" };

  // the names of the variables placed in the viewstate
  static readonly String VS_CURRENT_SORT_EXPRESSION = "currentSortExpression";
  static readonly String VS_CURRENT_SORT_ORDER = "currentSortOrder";

  ///*****************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event. It
  /// is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  private void Page_Load(object sender, System.EventArgs e)
  {
   String defaultSortExpression;
   enuSortOrder defaultSortOrder;

   if (!Page.IsPostBack)
   {
    // sort by title, ascending as the default
     defaultSortExpression = sortExpression[0];
    defaultSortOrder = enuSortOrder.soAscending;

    // bind data to the DataGrid
    this.ViewState.Add(VS_CURRENT_SORT_EXPRESSION, defaultSortExpression);
    this.ViewState.Add(VS_CURRENT_SORT_ORDER, defaultSortOrder);
    bindData(defaultSortExpression,
     defaultSortOrder);
   }
  } // Page_Load

  ///*****************************************************************
  /// <summary>
  /// This routine provides the event handler for the datagrid sort event.
  /// It is responsible re-binding the data to the datagrid by the selected
  /// column.
  /// </summary>
  ///
  /// <param name="source">Set to the source of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void dgBooks_SortCommand(Object source,
    System.Web.UI.WebControls.DataGridSortCommandEventArgs e)
  {
   String newSortExpression = null;
   String currentSortExpression = null;
```

```
enuSortOrder currentSortOrder;

// get the current sort expression and order from the viewstate
currentSortExpression =
 (String)(this.ViewState[VS_CURRENT_SORT_EXPRESSION]);
currentSortOrder =
 (enuSortOrder)(this.ViewState[VS_CURRENT_SORT_ORDER]);
// check to see if this is a new column or the sort order
// of the current column needs to be changed.
newSortExpression = e.SortExpression;
if (newSortExpression == currentSortExpression)
{
 // sort column is the same so change the sort order
 if (currentSortOrder == enuSortOrder.soAscending)
 {
 currentSortOrder = enuSortOrder.soDescending;
 }
 else
 {
 currentSortOrder = enuSortOrder.soAscending;
 }
}
else
{
  // sort column is different so set the new column with ascending
  // sort order
  currentSortExpression = newSortExpression;
  currentSortOrder = enuSortOrder.soAscending;
}

// update the view state with the new sort information
this.ViewState.Add(VS_CURRENT_SORT_EXPRESSION, currentSortExpression);
this.ViewState.Add(VS_CURRENT_SORT_ORDER, currentSortOrder);

   // rebind the data in the datagrid
   bindData(currentSortExpression,
  currentSortOrder);
 } // dgBooks_SortCommand

 ///*****************************************************************
 /// <summary>
 /// This routine provides the event handler for the page index changed
 /// event of the datagrid. It is responsible for setting the page index
 /// from the passed arguments and rebinding the data.
 /// </summary>
 ///
 /// <param name="source">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>

 protected void dgBooks_PageIndexChanged(Object source,
     System.Web.UI.WebControls.DataGridPageChangedEventArgs e)
 {
  String currentSortExpression;
   enuSortOrder currentSortOrder;

   // set new page index and rebind the data
   dgBooks.CurrentPageIndex = e.NewPageIndex;

   // get the current sort expression and order from the viewstate
   currentSortExpression =
 (String)(this.ViewState[VS_CURRENT_SORT_EXPRESSION]);
   currentSortOrder =
   (enuSortOrder)(this.ViewState[VS_CURRENT_SORT_ORDER]);

   // rebind the data in the datagrid
   bindData(currentSortExpression,
  currentSortOrder);
 } // dgCustomers_PageIndexChanged

 ///*****************************************************************
 /// <summary>
 /// This routine queries the database for the data to displayed and binds
 /// it to the datagrid
 /// </summary>
 ///
 /// <param name="sortExpression">Set to the sort expression to use for
 ///       sorting the data</param>
 /// <param name="sortOrder">Set to the requried sort order</param>
 private void bindData(String sortExpression,
     enuSortOrder sortOrder)
 {
   OleDbConnection dbConn = null;
   OleDbDataAdapter da = null;
   DataTable dTable = null;
   String strConnection = null;
   String strSQL = null;
   int index = 0;
   DataGridColumn col = null;
   String colImage = null;
   String strSortOrder = null;

   try
   {
   // get the connection string from web.config and open a connection
   // to the database
   strConnection = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
   dbConn = new OleDbConnection(strConnection);
   dbConn.Open();

   // build the query string and get the data from the database
   if (sortOrder == enuSortOrder.soAscending)
   {
     strSortOrder = " ASC";
```

```
      }
      else
      {
        strSortOrder = " DESC";
      }

      strSQL = "SELECT Title, ISBN, Publisher " +
     "FROM Book " +
     "ORDER BY " + sortExpression +
     strSortOrder;

      da = new OleDbDataAdapter(strSQL, dbConn);
      dTable = new DataTable();
      da.Fill(dTable);

      // loop through the columns in the datagrid updating the heading to
      // mark which column is the sort column and the sort order
       for (index = 0; index < dgBooks.Columns.Count; index++)
      {
        col = dgBooks.Columns[index];
     // check to see if this is the sort column
     if (col.SortExpression == sortExpression)
     {
       // this is the sort column so determine whether the ascending or
       // descending image needs to be included
       if (sortOrder == enuSortOrder.soAscending)
       {
          colImage = " <img src='images/sort_ascending.gif' border='0'>";
       }
       else
       {
          colImage = " <img src='images/sort_descending.gif' border='0'>";
       }
     }
     else
     {
       // This is not the sort column so include no image html
       colImage = "";
     } // if (col.SortExpression == sortExpression)

     // set the title for the column
     col.HeaderText = columnTitle[index] + colImage;
      } // for index

      // set the source of the data for the datagrid control and bind it
      dgBooks.DataSource = dTable;
      dgBooks.DataBind();
    } // try

    finally
    {
   //clean up
```

```
      if (dbConn != null)
   {
   dbConn.Close();
    }
   } // finally
 } // bindData
  }  //  CH02DatagridWithSortingAndPagingCS
}
```

PREV

Recipe 2.13. Paging Through a Record-Heavy DataGrid

Problem

You need to display a large set of data in a `DataGrid` , yet the user must be able to page through it quickly. This approach is beneficial anytime you have to navigate through thousands of records.

Solution

Use custom paging with a `DataGrid` and, using a stored procedure, read from the database only the data that is needed for a given page. An example of the output that can be achieved with this approach is shown in Figure 2-13. Examples 2-33 , 2-34 through 2-35 show the *.aspx* and code-behind files for an application that illustrates this approach; the application uses the stored procedur
shown in Example 2-32 to retrieve the data to display.

Discussion

The solution we advocate for the problem of paging throughlarge datasets requires a somewhat different approach to custom paging and managing the labels used to display the current and total pages.

Figure 2-13. Paging through a record-heavy DataGrid output

To enable paging and to allow you to control movement within the dataset when data binding, the DataGrid's AllowPaging and AllowCustomPaging attributes must be set to TRue . When AllowCustomPaging is set to False (the default), the DataGrid assumes that all of the data that can be displayed in all pages is present in the data source, and it calculates the group of records to display from the CurrentPageIndex and PageSize attributes. When AllowCustomPaging is set to TRue , the DataGrid expects only one page (as defined by the PageSize attribute) to be present in the data source and you are responsible for filling the data source with the proper page of data.

```
<asp:DataGrid ID="dgBooks" runat="server"
     BorderColor="#000080"
     BorderWidth="2px"
    AutoGenerateColumns="False"
    Width="90%"
    HorizontalAlign="Center"
    AllowPaging="True"
    AllowCustomPaging="True"
    PageSize="10"
    PagerStyle-Visible="False">
```

For this example, the internal paging controls of theDataGrid are not used, so the PagerStyle-Visible attribute is set False to hide the DataGrid's pager control.

A pair of labels is used to display the current page and total number of pages available. In addition, four buttons are used to provide navigation (First, Prev, Next, and Last).

If you want to use the internal paging functionality with custom paging, the VirtualItemCount attribute must be set to the total number of items that can be displayed in the DataGrid (all pages). In addition, the CurrentPageIndex attribute must be set to the currently displayed page.

The code-behind uses two private variables to store the current page and total number of pages usec throughout the class. In the Page_Load event handler, the currentPage variable is initialized to 0 when the page is initially loaded, and then thebindData method is called to populate the DataGrid . When the page is being posted back, thecurrentPage and totalPages variables are set from the values in the labels used to display the information to the user. The data binding is then done, as required, by the specific event handlers.

Four event handler routines are included in the code-behind to handle the click events for the four buttons. The event handlers alter the currentPage variable as appropriate and rebind the data. To improve performance, the event handlers first check to see if the page needs changing and rebinding

With standard paging, all of the data is returned, even if there are thousands of rows, and the DataGrid determines which ones are displayed. In this case, however, thebindData method uses the stored procedure shown in Example 2-32 to retrieve only the data to be displayed for the required page.

The stored procedure uses three parameters: `pageNumber`, `pageSize`, and `totalRecords`. The `pageNumber` is an input parameter that defines the page to be displayed. The `pageSize` is an input parameter that defines the number of rows to be displayed per page; this must be the same as the `DataGrid's PageSize` property. `totalRecords` is an output parameter used to obtain the total number of rows of data available for display.

The stored procedure first calculates the index of the first record and the last record to display in the requested page, as shown here:

```
SELECT @firstRecordInPage = @pageNumber * @pageSize + 1
SELECT @lastRecordInPage = @firstRecordInPage + @pageSize
```

A temporary table is then created to store the data from the Book table in the desired order. This table contains an Identity column that is set up to number the records from 1 to the total number of records added to the table. This provides the ability to select only the specific rows needed for the requested page number.

```
CREATE TABLE #Book
(
 [ID] [int] IDENTITY (1, 1) NOT NULL ,
 [BookID] [int] NOT NULL ,
 [Title] [nvarchar] (100) NOT NULL ,
 [ISBN] [nvarchar] (50) NOT NULL ,
 [Publisher] [nvarchar] (50) NOT NULL
)
```

Next, the data from the Book table is copied into the temporary table and ordered by the book title. Now you have an ordered list of the books with the ID column set to `1` for the first book and N for the last book.

```
INSERT INTO #Book
(BookID, Title, ISBN, Publisher)
SELECT BookID, Title, ISBN, Publisher FROM Book ORDER BY Title
```

The next step is to query the temporary table for only the rows required for the page being displayed This is done by qualifying the query based on the ID being within the range of the first and last records to display.

```
SELECT * FROM #Book
WHERE ID >= @firstRecordInPage
AND ID < @lastRecordInPage
```

Finally, you query the Book table for the total number of books and set the `totalRecords` output parameter to the count:

```
SELECT @totalRecords = COUNT(*) FROM Book
```

The stored procedure used here was kept simple to illustrate the concept of returning only the required data. One negative of the example code is that all of the data from the Book table is copied to the temporary table, unnecessarily bloating the table. One way to reduce the amount of data copied is to copy rows only up to the last row required, a modification you will want to consider when adapting this code to your unique environment.

If you are using SQL Server 2005, you may want to consider implementing the stored procedure in VB.NET or C# and use the new `ExecutePagedReader` method in the `SqlCommand` class. The `Execute-PagedReader` method provides the ability to retrieve only the data you need in an efficient manner.

The `bindData` method first opens a connection to the database. A command is then created to execute the stored procedure, and the three parameters required for the stored procedure are added to it. The command is then executed using the `ExecuteReader` method and the returned data reader is set as the data source for the `DataGrid` .

The returned `DataReader` must be closed to retrieve the output parameter from the stored procedure. Attempting to access the output parameter before the `DataReader` is closed will return `null` .

Finally, the total number of records is retrieved from the parameter collection, and the labels on the form used to inform the user of the current page and total number of pages are initialized.

See Also

Recipe 2.8

Example 2-32. Stored procedure for record-heavy DataGrid

```
CREATE PROCEDURE getPageData
@pageNumber INT,
@pageSize INT,
@totalRecords INT OUTPUT
AS
DECLARE @firstRecordInPage INT
DECLARE @lastRecordInPage INT

-- Calculate the number of rows needed to get to the current page
SELECT @firstRecordInPage = @pageNumber * @pageSize + 1
SELECT @lastRecordInPage = @firstRecordInPage + @pageSize

-- Create a temporary table to copy the book data into.
-- Include only the columns needed with an additional ID
-- column that is the primary key of the temporary table.
-- In addition, it is an identity that will number the
-- records copied into the table starting with 1 thus allowing
-- us to query only for the specific records needed for the
-- requested page.
CREATE TABLE #Book
(
[ID] [int] IDENTITY (1, 1) NOT NULL ,
[BookID] [int] NOT NULL ,
[Title] [nvarchar] (100) NOT NULL ,
[ISBN] [nvarchar] (50) NOT NULL ,
[Publisher] [nvarchar] (50) NOT NULL
)

-- Copy the data from the book table into the temp table
INSERT INTO #Book
(BookID, Title, ISBN, Publisher)
SELECT BookID, Title, ISBN, Publisher FROM Book ORDER BY Title

-- Get the rows required for the passed page
SELECT * FROM #Book
WHERE ID >= @firstRecordInPage
AND ID < @lastRecordInPage

-- Get the total number of records in the table
SELECT @totalRecords = COUNT(*) FROM Book
GO
```

Example 2-33. Paging through a record-heavy DataGrid (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
```

```
    CodeFile="CH02 LargeDatasetPagingVB1.aspx.vb"
     Inherits="ASPNetCookbook.VBExamples.CH02LargeDatasetPagingVB1"
    Title="DataGrid With  Large Data Set Paging" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">
    DataGrid With Large Data Set Paging (VB)
  </div>
  <asp:DataGrid ID="dgBooks" runat="server"
    BorderColor="#000080"
    BorderWidth="2px"
    AutoGenerateColumns="False"
    Width="90%"
    HorizontalAlign="Center"
    AllowPaging="True"
    AllowCustomPaging="True"
    PageSize="10"
    PagerStyle-Visible="False">
  <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
  <ItemStyle CssClass="tableCellNormal" />
  <AlternatingItemStyle CssClass="tableCellAlternating" />
  <Columns>
 <asp:BoundColumn HeaderText="Title" DataField="Title"/>
 <asp:BoundColumn HeaderText="ISBN" DataField="ISBN"
    ItemStyle-HorizontalAlign="Center"/>
 <asp:BoundColumn HeaderText="Publisher" DataField="Publisher"
    ItemStyle-HorizontalAlign="Center"/>
  </Columns>
</asp:DataGrid>
<table width="40%" border="0" align="center">
  <tr>
     <td colspan="4" align="center">
   Displaying page
   <asp:Literal id="labCurrentPage" runat="server" /> of
   <asp:Literal id="labTotalPages" runat="server" /></td>
 </tr>
 <tr>
   <td align="center">
     <input id="btnFirst" runat="server"
    type="button"
    value="First"
    onserverclick="btnFirst_ServerClick"/>
    </td>
 <td align="center">
      <input id="btnPrev" runat="server"
    type="button"
    value="Prev"
    onserverclick="btnPrev_ServerClick"/>
    </td>
    <td align="center">
      <input id="btnNext" runat="server"
         type="button"
       value="Next"
      onserverclick="btnNext_ServerClick"/>
    </td>
    <td align="center">
      <input id="Last" runat="server"
    type="button"
    value="Last"
    onserverclick="btnLast_ServerClick"/>
   </td>
    </tr>
  </table>
</asp:Content>
```

Example 2-34. Paging through a record-heavy DataGrid (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' CH02 LargeDatasetPagingVB1.aspx
 ''' </summary>
 Partial Class CH02LargeDatasetPagingVB1
  Inherits System.Web.UI.Page

  'private variables used to store the current page and total number of
  'pages. This is required since the CurrentPageIndex and PageCount
  'properties of the datagrid cannot be used with custom paging

  Private currentPage As Integer
  Private totalPages As Integer

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
   If (Page.IsPostBack) Then
  'This is a post back so initialize the current and total page variables
  'with the values currently being displayed
  currentPage = CInt(labCurrentPage.Text) - 1 'zero based page numbers
  totalPages = CInt(labTotalPages.Text)
   Else
     'This is the first rendering of the form so set the current page to the
  'first page and bind the data
  currentPage = 0
  bindData()
   End If
 End Sub 'Page_Load

 '''**************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the first button click
 ''' event. It is responsible for setting the page index to the first
 ''' page and rebinding the data.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnFirst_ServerClick(ByVal sender As Object, _
           ByVal e As System.EventArgs)
   'set new page index and rebind the data
   If (currentPage > 0) Then
  currentPage = 0
  bindData()
   End If
 End Sub 'btnFirst_ServerClick

 '''**************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the previous button click
 ''' event. It is responsible for setting the page index to the previous
 ''' page and rebinding the data.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnPrev_ServerClick(ByVal sender As Object, _
          ByVal e As System.EventArgs)
   'set new page index and rebind the data
   If (currentPage > 0) Then
     currentPage -= 1
  bindData()
   End If
 End Sub 'btnPrev_ServerClick

 '''**************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the next button click
 ''' event. It is responsible for setting the page index to the next
 ''' page and rebinding the data.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnNext_ServerClick(ByVal sender As Object, _
          ByVal e As System.EventArgs)
   'set new page index and rebind the data
   If (currentPage < totalPages - 1) Then
  currentPage += 1
  bindData()
   End If
 End Sub 'btnNext_ServerClick

 '''**************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the last button click
 ''' event. It is responsible for setting the page index to the last
 ''' page and rebinding the data.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnLast_ServerClick(ByVal sender As Object, _
          ByVal e As System.EventArgs)
   'set new page index and rebind the data
   If (currentPage < totalPages - 1) Then
  currentPage = totalPages - 1
  bindData()
   End If
 End Sub 'btnLast_ServerClick

 '''**************************************************************************
 ''' <summary>
 ''' This routine queries the database for the data to displayed and binds
 ''' it to the datagrid
 ''' </summary>
 Private Sub bindData()
   Dim dbConn As OleDbConnection = Nothing
   Dim dCmd As OleDbCommand = Nothing
   Dim dReader As OleDbDataReader = Nothing
   Dim param As OleDbParameter = Nothing
   Dim strConnection As String
   Dim strSQL As String
   Dim totalRecords As Integer

   Try
  'get the connection string from web.config and open a connection
  'to the database
  strConnection = ConfigurationManager. _
   ConnectionStrings("dbConnectionString").ConnectionString
  dbConn = New OleDb.OleDbConnection(strConnection)
```

```
  dbConn.Open()

  'create command to execute the stored procedure along with the
  'parameters required in and out of the procedure
  strSQL = "getPageData"   'name of stored procedure
  dCmd = New OleDbCommand(strSQL, dbConn)
  dCmd.CommandType = CommandType.StoredProcedure
  param = dCmd.Parameters.Add("pageNumber", OleDbType.Integer)
  param.Direction = ParameterDirection.Input
  param.Value = currentPage

  param = dCmd.Parameters.Add("pageSize", OleDbType.Integer)
  param.Direction = ParameterDirection.Input
  param.Value = dgBooks.PageSize

  param = dCmd.Parameters.Add("totalRecords", OleDbType.Integer)
  param.Direction = ParameterDirection.Output

  'execute the stored procedure and set the datasource for the datagrid
  dReader = dCmd.ExecuteReader()
  dgBooks.DataSource = dReader
  dgBooks.DataBind()

  'close the dataReader to make the output parameter available
  dReader.Close()

  'output information about the current page and total number of pages
  totalRecords = CInt(dCmd.Parameters.Item("totalRecords").Value)
  totalPages = CInt(Math.Ceiling(totalRecords / dgBooks.PageSize))
  labTotalPages.Text = totalPages.ToString()
  labCurrentPage.Text = (currentPage + 1).ToString()

  Finally
    'cleanup
    If (Not IsNothing(dReader)) Then
      dReader.Close()
    End If

    If (Not IsNothing(dbConn)) Then
    dbConn.Close()
    End If
    End Try
 End Sub 'bindData
  End Class 'CH02 LargeDatasetPagingVB1
End Namespace
```

Example 2-35. Paging through a record-heavy DataGrid (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
  ///***************************************************************************
  /// <summary>
  /// This class provides the code behind for
  /// CH02LargeDatasetPagingCS1.aspx
  /// </summary>
  public partial class CH02 LargeDatasetPagingCS1 : System.Web.UI.Page
  {

  // private variables used to store the current page and total number of
  // pages. This is required since the CurrentPageIndex and PageCount
  // properties of the datagrid cannot be used with custom paging
  private int currentPage;
  private int totalPages;

   ///*************************************************************************
   /// <summary>
   /// This routine provides the event handler for the page load event.
   /// It is responsible for initializing the controls on the page.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void Page_Load(object sender, EventArgs e)
   {
     if (Page.IsPostBack)
  {
    // This is a post back so initialize the current and total page
    // variables with the values currently being displayed
    currentPage = Convert.ToInt32(labCurrentPage.Text) - 1;
    totalPages = Convert.ToInt32(labTotalPages.Text);
  }
  else
  {
    // This is the first rendering of the form so set the current page to
     // the first page and bind the data
    currentPage = 0;
    bindData();
  }
   } // Page_Load

   ///*************************************************************************
   /// <summary>
   /// This routine provides the event handler for the first button click
   /// event. It is responsible for setting the page index to the first
   /// page and rebinding the data.
```

```
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>

 protected void btnFirst_ServerClick(Object sender,
       System.EventArgs e)
    {
   // set new page index and rebind the data
if (currentPage > 0)
{
  currentPage = 0;
  bindData();
}
 } // btnFirst_ServerClick

  ///**************************************************************************
 /// <summary>
 /// This routine provides the event handler for the previous button click
 /// event. It is responsible for setting the page index to the previous
 /// page and rebinding the data.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnPrev_ServerClick(Object sender,
                System.EventArgs e)
 {
   // set new page index and rebind the data
if (currentPage > 0)
{
  currentPage -= 1;
  bindData();
}
 } // btnPrev_ServerClick

  ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the next button click
 /// event. It is responsible for setting the page index to the next
 /// page and rebinding the data.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnNext_ServerClick(Object sender,
      System.EventArgs e)
 {
   // set new page index and rebind the data
if (currentPage < totalPages - 1)
{
  currentPage += 1;
  bindData();
}
 } // btnNext_ServerClick

 ///***************************************************************************
 /// <summary>
 /// This routine provides the event handler for the last button click
 /// event. It is responsible for setting the page index to the last
 /// page and rebinding the data.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnLast_ServerClick(Object sender,
      System.EventArgs e)
 {
   // set new page index and rebind the data
if (currentPage < totalPages - 1)
{
  currentPage = totalPages - 1;
  bindData();
}
 } // btnLast_ServerClick

 ///**********************************************************************
 /// <summary>
 /// This routine queries the database for the data to displayed and binds
 /// it to the datagrid
 /// </summary>
 private void bindData()
 {
   OleDbConnection dbConn = null;
OleDbCommand dCmd = null;
OleDbDataReader dReader = null;
OleDbParameter param = null;
String strConnection = null;
String strSQL = null;
int totalRecords = 0;

try
{
  // get the connection string from web.config and open a connection
  // to the database
  strConnection = ConfigurationManager.
   ConnectionStrings["dbConnectionString"].ConnectionString;
  dbConn = new OleDbConnection(strConnection);
  dbConn.Open();

  // create command to execute the stored procedure along with the
  // parameters required in/out of the procedure
  strSQL = "getPageData"; // name of stored procedure
  dCmd = new OleDbCommand(strSQL, dbConn);
```

```
     dCmd.CommandType = CommandType.StoredProcedure;

     param = dCmd.Parameters.Add("pageNumber", OleDbType.Integer);
     param.Direction = ParameterDirection.Input;
     param.Value = currentPage;

     param = dCmd.Parameters.Add("pageSize", OleDbType.Integer);
     param.Direction = ParameterDirection.Input;
     param.Value = dgBooks.PageSize;

     param = dCmd.Parameters.Add("totalRecords", OleDbType.Integer);
     param.Direction = ParameterDirection.Output;
     //execute the stored procedure and set the datasource for the datagrid
     dReader = dCmd.ExecuteReader();
     dgBooks.DataSource = dReader;
     dgBooks.DataBind();

     // close the dataReader to make the output parameter available
     dReader.Close();

     // output information about the current page and total number of pages
     totalRecords = Convert.ToInt32(dCmd.Parameters["totalRecords"].Value);
     totalPages = Convert.ToInt32(Math.Ceiling(Convert.ToDouble(totalRecords) /
      dgBooks.PageSize));
     labTotalPages.Text = totalPages.ToString();
     labCurrentPage.Text = (currentPage + 1).ToString();
    } // try

    finally
    {
    //clean up
    if (dReader != null)
    {
     dReader.Close();
    }
    if (dbConn != null)
    {
     dbConn.Close();
    }
    } // finally
     } // bindData
  } // CH02LargeDatasetPagingCS1
}
```

Recipe 2.14. Editing Data Within a DataGrid

Problem

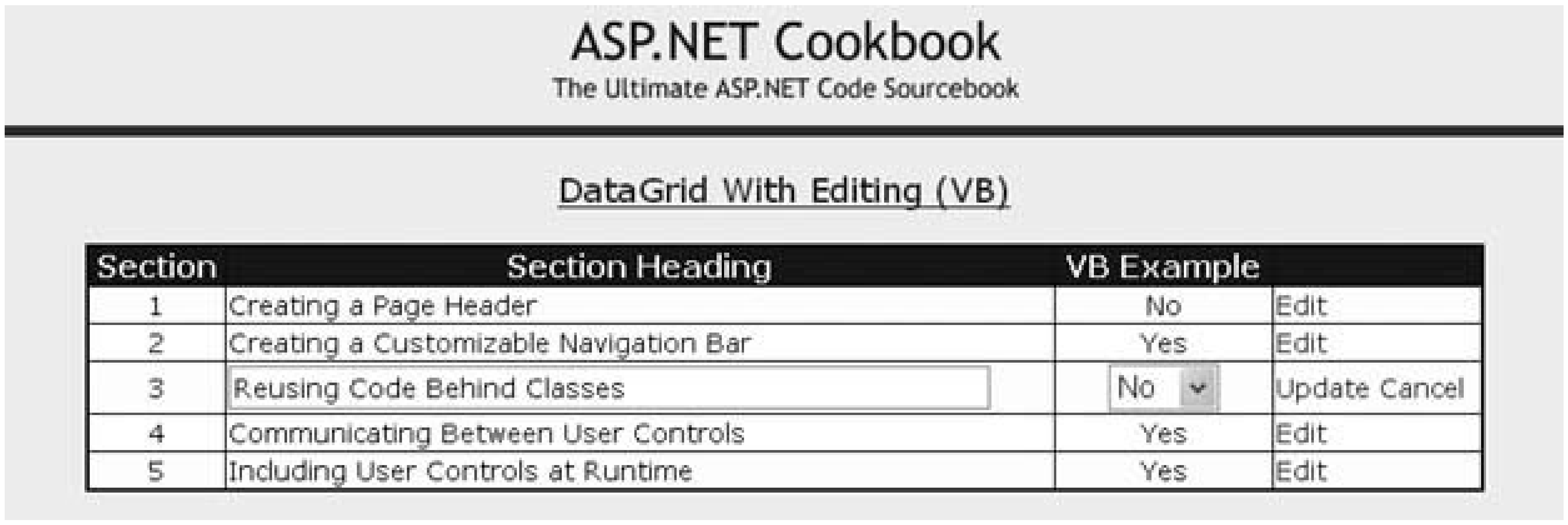
You want to allow the user to edit the data within the table displayed by a`DataGrid` .

Solution

Add an `EditCommandColumn` column type to the `DataGrid` control's display to enable editing of the data fields of each record. A typical example of normal display mode output is shown in Figure 2-14, and an example of edit mode output is shown in Figure 2-15. Examples 2-36 , 2-37 through 2-38 show the *.aspx* and code-behind files for the application that produces this result.

Figure 2-14. DataGrid with editingnormal mode

Figure 2-15. DataGrid with editingrow edit mode


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
DataGrid With Editing (VB)

Section	Section Heading	VB Example	
1	Creating a Page Header	No	Edit
2	Creating a Customizable Navigation Bar	Yes	Edit
3	Reusing Code Behind Classes	No	Update Cancel
4	Communicating Between User Controls	Yes	Edit
5	Including User Controls at Runtime	Yes	Edit

Discussion

This recipe uses the built-in editing facilities of the DataGrid control, in particular the EditCommandColumn column type, which provides Edit command buttons for editing data items in each row of a DataGrid . The EditText, CancelText , and UpdateText properties define the text to be output for the Edit command button's Edit, Cancel, and Update hyperlinks, respectively.

```
<asp:EditCommandColumn ButtonType="LinkButton"
     EditText="Edit"
     CancelText="Cancel"
     UpdateText="Update" />
```

The ButtonType attribute defines the type of button to output. You can specify LinkButton , which provides hyperlinked text, or PushButton , which outputs an HTML button.

> The Edit command button's EditText, CancelText , and UpdateText properties can be set to HTML. For example, to output an image for the links, you can use `<img src="images/buttons/editButton.gif" border="0" alt="Edit"/>` .

In our example that implements this solution, three columns are defined for the DataGrid . The first uses an asp:BoundColumn element with the ReadOnly attribute set to true to prevent users from editing the field contents:

```
<asp:BoundColumn DataField="SectionNumber"
   ItemStyle-HorizontalAlign="Center"
   HeaderText="Section"
   ReadOnly="True" />
```

The second column uses an asp:TemplateColumn element to define a layout template for normal display (ItemTemplate) and edit mode display (EditItemTemplate) . The EditItemTemplate property defines an asp:TextBox control to control the size and other aspects of the field contents. Both templates are bound to the "SectionHeading" data.

```
<asp:TemplateColumn HeaderText="Section Heading">
 <ItemTemplate>
  <%#Eval("SectionHeading")%>
 </ItemTemplate>
 <EditItemTemplate>
  <asp:TextBox id="txtSectionHeading" runat="server"
           Columns="55" cssClass="tableCellNormal"
     text='<%# Eval("SectionHeading") %>' />
 </EditItemTemplate>
</asp:TemplateColumn>
```

Like the second column, the third column uses anasp:TemplateColumn element. In this case, however, the EditItemTemplate property defines an asp:DropDownList control, allowing the user to select only from valid choices for the column. This column is bound to theyesNoSelections ArrayList created in the code-behind. The selection in the drop-down list is initialized to the current value in th database by binding the SelectedIndex to the index of the value in the ArrayList .

```
<asp:TemplateColumn HeaderText="VB Example"
      ItemStyle-HorizontalAlign="Center">
     <ItemTemplate>
  <%#yesNoSelections.Item(CInt(Eval("HasVBExample")))%>
  </ItemTemplate>
  <EditItemTemplate>
  <asp:DropDownList id="selHasVBSample" runat="server"
   DataSource="<%# yesNoSelections %>"
   DataTextField="Text"
   DataValueField="Value"
   SelectedIndex='<%# CInt(Eval("HasVBExample")) %>' />
 </EditItemTemplate>
</asp:TemplateColumn>
```

The Protected yesNoSelections As ArrayList declaration is added at the class level in the code-behind to provide access to the ArrayList from the code in the *.aspx* file.

Page_Load just calls bindData , as is typical in this chapter's recipes. However, bindData is different from the norm in two ways. First, the ArrayList is built with the selections that are applicable for the user to select from when changing the value of the Has VB Example column. Second, the line dgProblems.DataKeyField = "EditProblemID" is added to have the DataGrid maintain the primary key value for each row without having to add it to the grid as a column (hidden or visible). This approach stores the primary key value for each row in the view state only so it can be recovered when needed on the server side. It has the advantage of hiding the value from prying eyes.

The dgProblems_EditCommand method handles the event generated when the user clicks the Edit link within a row. It sets the EditItemIndex to the selected row, which causes ASP.NET to use the Edit Template when the data for the row is rebound along with the Cancel and Update links in the edit command column.

The dgProblems_CancelCommand method handles the event generated when the user clicks the Cancel link in the row being edited. It sets the EditItemIndex to 1 to display the DataGrid in normal mode when the data is rebound.

The dgProblems_UpdateCommand method handles the event generated when the user clicks the Update link in the row being edited. It extracts the edited data, updates the data in the database, and resets the DataGrid to normal mode when the data is rebound (see comments in the code for more details).

Example 2-36. DataGrid with editing (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02DataGridWithEditingVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02DataGridWithEditingVB"
 Title="DataGrid With Editing" %>

<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  DataGrid With Editing (VB)
 </div>
 <asp:DataGrid id="dgProblems" runat="server"
     BorderColor="#000080" BorderWidth="2px"
     AutoGenerateColumns="False"
     HorizontalAlign="center"
     Width="90%"
     OnCancelCommand="dgProblems_CancelCommand"
     OnEditCommand="dgProblems_EditCommand"
     OnUpdateCommand="dgProblems_UpdateCommand">
  <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
  <ItemStyle cssClass="tableCellNormal" />
  <AlternatingItemStyle cssClass="tableCellAlternating" />
  <Columns>
   <asp:BoundColumn DataField="SectionNumber"
     ItemStyle-HorizontalAlign="Center"
     HeaderText="Section"
     ReadOnly="True" />
  <asp:TemplateColumn HeaderText="Section Heading">
   <ItemTemplate>
    <%#Eval("SectionHeading")%>
   </ItemTemplate>
   <EditItemTemplate>
    <asp:TextBox id="txtSectionHeading" runat="server"
     Columns="55" cssClass="tableCellNormal"
     text='<%# Eval("SectionHeading") %>' />
   </EditItemTemplate>
  </asp:TemplateColumn>

  <asp:TemplateColumn HeaderText="VB Example"
       ItemStyle-HorizontalAlign="Center">
        <ItemTemplate>
     <%#yesNoSelections.Item(CInt(Eval("HasVBExample")))%>
     </ItemTemplate>
     <EditItemTemplate>
      <asp:DropDownList id="selHasVBSample" runat="server"
       DataSource="<%# yesNoSelections %>"
       DataTextField="Text"
       DataValueField="Value"
     SelectedIndex='<%# CInt(Eval("HasVBExample")) %>' />
   </EditItemTemplate>
  </asp:TemplateColumn>

  <asp:EditCommandColumn ButtonType="LinkButton"
       EditText="Edit"
       CancelText="Cancel"
       UpdateText="Update" />
  </Columns>
 </asp:DataGrid>
</asp:Content>
```

Example 2-37. DataGrid with editing code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02DataGridWithEditingVB.aspx
 ''' </summary>
 Partial  Class  CH02DataGridWithEditingVB
  Inherits System.Web.UI.Page
  'The following variable contains the list of yes/no selections used in
  'the dropdown lists and is declared protected to provide access to the
  'data from the aspx page
  Protected yesNoSelections As ArrayList

  '''*******************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
      If (Not Page.IsPostBack) Then
       bindData()
      End If
  End Sub 'Page_Load

  '''******************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the cancel command click
  ''' event. It is responsible for resetting the edit item index to no item
  ''' and rebinding the data.
  ''' </summary>
  '''
  ''' <param name="source">Set to the source of the event</param>
  ''' <param name="e">Set to the event arguments</param>
```
`Protected Sub dgProblems_CancelCommand(ByVal source As Object, _`
`  ByVal e As System.Web.UI.WebControls.DataGridCommandEventArgs)`
` dgProblems.EditItemIndex = -1`
` bindData()`
`End Sub 'dgProblems_CancelCommand`

```
'''******************************************************************
''' <summary>
''' This routine provides the event handler for the edit command click
''' event. It is responsible for setting the edit item index to the
''' selected item and rebinding the data.
''' </summary>
'''
''' <param name="source">Set to the source of the event</param>
''' <param name="e">Set to the event arguments</param>
```
`Protected Sub dgProblems_EditCommand(ByVal source As Object, _`
`  ByVal e As System.Web.UI.WebControls.DataGridCommandEventArgs)`
` dgProblems.EditItemIndex = e.Item.ItemIndex`
` bindData()`
`End Sub 'dgProblems_EditCommand`

```
'''******************************************************************
   ''' <summary>
''' This routine provides the event handler for the update command click
''' event. It is responsible for updating the contents of the database
''' with the date entered for the item currently being edited.
''' </summary>
'''
''' <param name="source">Set to the source of the event</param>
''' <param name="e">Set to the event arguments</param>
```

```
Protected Sub dgProblems_UpdateCommand(ByVal source As Object, _
  ByVal e As System.Web.UI.WebControls.DataGridCommandEventArgs)
 Dim dbConn As OleDbConnection = Nothing
 Dim dCmd As OleDbCommand = Nothing
 Dim sectionHeading As String
 Dim hasVBSample As Integer
 Dim strConnection As String
 Dim strSQL As String
 Dim rowsAffected As Integer

 Try
  'get the edited section heading and "has vb sample" data
  'NOTE: This can be done by using the FindControl method of the edited
  '    item because EditItemTemplates were used and the controls in the
  '    templates where given IDs. If a standard BoundColumn was used,
  '    the data would have to be acccessed using the cells collection
  '    (e.g. e.Item.Cells(0).Text would access the section number
  '    column in this example.
  sectionHeading = CType(e.Item.FindControl("txtSectionHeading"), _
        TextBox).Text( )
  hasVBSample = CInt(CType(e.Item.FindControl("selHasVBSample"), _
      DropDownList).SelectedItem.Value)
  'get the connection string from web.config and open a connection
  'to the database
  strConnection = ConfigurationManager. _
  ConnectionStrings("dbConnectionString").ConnectionString
  dbConn = New OleDbConnection(strConnection)
  dbConn.Open( )

  'update data in database
  'NOTE: The primary key used to uniquely identify the row being edited
  '  is accessed through the DataKeys collection of the DataGrid.
  strSQL = "UPDATE EditProblem " & _
   "SET SectionHeading=?" & _
   ",HasVBExample=?" & _
   " WHERE EditProblemID=" & _
   dgProblems.DataKeys(e.Item.ItemIndex).ToString( )
  dCmd = New OleDbCommand(strSQL, dbConn)
  dCmd.Parameters.Add(New OleDbParameter("SectionHeading", sectionHeading))
  dCmd.Parameters.Add(New OleDbParameter("HasVBSample", hasVBSample))
  rowsAffected = dCmd.ExecuteNonQuery( )

  'TODO: production code should check the number of rows affected here to
  'make sure it is exactly 1 and output the appropriate success or
  'failure information to the user.
  'reset the edit item and rebind the data
  dgProblems.EditItemIndex = -1
  bindData( )

 Finally
  'cleanup
   If (Not IsNothing(dbConn)) Then
  dbConn.Close( )
   End If
  End Try
   End Sub 'dgProblems_UpdateCommand

   '''*************************************************************
   ''' <summary>
   ''' This routine queries the database for the data to displayed and
   ''' binds it to the DataGrid.
   ''' </summary>
   Private Sub bindData( )
  Dim dbConn As OleDbConnection = Nothing
  Dim da As OleDbDataAdapter = Nothing
  Dim dTable As DataTable = Nothing
  Dim strConnection As String
  Dim strSQL As String

  Try
  'get the connection string from web.config and open a connection
  'to the database
  strConnection = ConfigurationManager. _
  ConnectionStrings("dbConnectionString").ConnectionString
  dbConn = New OleDbConnection(strConnection)
  dbConn.Open( )

  'build the query string and get the data from the database
  strSQL = "SELECT EditProblemID, SectionNumber" & _
   ", SectionHeading, HasVBExample" & _
   " FROM EditProblem" & _
   " ORDER BY SectionNumber"

  da = New OleDbDataAdapter(strSQL, dbConn)
  dTable = New DataTable
  da.Fill(dTable)

  'build the arraylist with the acceptable responses to the
  '"Has VB Sample" field
  yesNoSelections = New ArrayList(2)
  yesNoSelections.Add(New ListItem("No", "0"))
  yesNoSelections.Add(New ListItem("Yes", "1")) \

  'set the source of the data for the datagrid control and bind it
  dgProblems.DataSource = dTable
  dgProblems.DataKeyField = "EditProblemID"
  dgProblems.DataBind( )

  Finally
  'cleanup
  If (Not IsNothing(dbConn)) Then
  dbConn.Close( )
  End If
 End Try
```

```
  End Sub 'bindData
 End Class 'CH02DataGridWithEditingVB
End Namespace
```

Example 2-38. DataGrid with editing code-behind (.cs)

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02DataGridWithEditingCS.aspx
 /// </summary>
 public partial class CH02DataGridWithEditingCS : System.Web.UI.Page
 {
  // The following variable contains the list of yes/no selections used in
  // the dropdown lists and is declared protected to provide access to the
  // data from the aspx page
  protected ArrayList yesNoSelections;

  ///*****************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   if (!Page.IsPostBack)
   {
    bindData( );
   }
  } // Page_Load

  ///*****************************************************************
  /// <summary>
  /// This routine provides the event handler for the cancel command click
  /// event. It is responsible for resetting the edit item index to no item
  /// and rebinding the data.
  /// </summary>
  ///
```

```
/// <param name="source">Set to the source of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void dgProblems_CancelCommand(Object source,
    System.Web.UI.WebControls.DataGridCommandEventArgs e)
   {
 dgProblems.EditItemIndex = -1;
 bindData( );
} // dgProblems_CancelCommand

 ///*******************************************************************
/// <summary>
/// This routine provides the event handler for the edit command click
/// event. It is responsible for setting the edit item index to the
/// selected item and rebinding the data.
/// </summary>
///
/// <param name="source">Set to the source of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void dgProblems_EditCommand(Object source,
    System.Web.UI.WebControls.DataGridCommandEventArgs e)
   {
 dgProblems.EditItemIndex = e.Item.ItemIndex;
 bindData( );
} // dgProblems_EditCommand

 ///*******************************************************************
/// <summary>
/// This routine provides the event handler for the update command click
/// event. It is responsible for updating the contents of the database
/// with the date entered for the item currently being edited.
/// </summary>
///
/// <param name="source">Set to the source of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void dgProblems_UpdateCommand(Object source,
    System.Web.UI.WebControls.DataGridCommandEventArgs e)
{
 OleDbConnection dbConn = null;
 OleDbCommand dCmd = null;
 String sectionHeading = null;
 int hasCSSample;
 String strConnection = null;
 String strSQL = null;
 int rowsAffected;
 DropDownList ddl = null;

 try
 {
  // get the edited section heading and "has vb sample" data
  // NOTE: This can be done by using the FindControl method of the edited
  //   item because EditItemTemplates were used and the controls in
  //   the templates where given IDs. If a standard BoundColumn was
//   used, the data would have to be acccessed using the cells
//   collection (e.g. e.Row.Cells(0).Text would access the section
//   number column in this example.
sectionHeading =
((TextBox)(e.Item.FindControl("txtSectionHeading"))).Text;
ddl = (DropDownList)(e.Item.FindControl("selHasCSSample"));
hasCSSample = Convert.ToInt32(ddl.SelectedItem.Value);

// get the connection string from web.config and open a connection
// to the database
strConnection = ConfigurationManager.
ConnectionStrings["dbConnectionString"].ConnectionString;
dbConn = new OleDbConnection(strConnection);
dbConn.Open( );

// update data in database
// NOTE: The primary key used to uniquely identify the row being edited
//   is accessed through the DataKeys collection of the DataGrid.
strSQL = "UPDATE EditProblem " +
 "SET SectionHeading='" + sectionHeading + "'" +
 ",HasCSExample=" + hasCSSample +
 " WHERE EditProblemID=" +
 dgProblems.DataKeys[e.Item.ItemIndex].ToString( );
 dCmd = new OleDbCommand(strSQL, dbConn);
 rowsAffected = dCmd.ExecuteNonQuery( );

 // TODO: production code should check the number of rows affected here to
 // make sure it is exactly 1 and output the appropriate success or
 // failure information to the user.

 // reset the edit item and rebind the data
 dgProblems.EditItemIndex = -1;
 bindData( );
 }

 finally
 {
//cleanup
if (dbConn != null)
{
dbConn.Close( );
}
 }
} // dgProblems_UpdateCommand

///****************************************************************
/// <summary>
/// This routine queries the database for the data to displayed and
/// binds it to the DataGrid.
/// </summary>
private void bindData( )
{
   OleDbConnection dbConn = null;
   OleDbDataAdapter da = null;
   DataTable dTable = null;
   String strConnection = null;
   String strSQL = null;

   try
   {
    // get the connection string from web.config and open a connection
    // to the database
    strConnection = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(strConnection);
    dbConn.Open( );

    // build the query string and get the data from the database
    strSQL = "SELECT EditProblemID, SectionNumber" +
     ", SectionHeading, HasCSExample " +
     "FROM EditProblem " +
     "ORDER BY SectionNumber";
     da = new OleDbDataAdapter(strSQL, dbConn);
     dTable = new DataTable( );
     da.Fill(dTable);

     // build the arraylist with the acceptable responses to the
     // "Has C# Sample" field
     yesNoSelections = new ArrayList(2);
     yesNoSelections.Add(new ListItem("No", "0"));
     yesNoSelections.Add(new ListItem("Yes", "1"));

     // set the source of the data for the datagrid control and bind it
     dgProblems.DataSource = dTable;
     dgProblems.DataKeyField = "EditProblemID";
     dgProblems.DataBind( );
   } // try

   finally
   {
    //clean up
    if (dbConn != null)
    {
    dbConn.Close( );
    }
   } // finally
  } // bindData
 } //  CH02DataGridWithEditingCS
}
```

PREV

Recipe 2.15. Navigating and Sorting Within a GridView

Problem

You want to use a `GridView` with sorting and paging.

Solution

Enable the sorting and paging functions of the`GridView` and add custom coding to display the sort order in the header of the current sort column. Figure 2-16shows a typical `GridView` with sorting and paging implemented. Examples 2-39 , 2-40 through 2-41 show the *.aspx* file and the code-behind files for an application that produces this output.

Figure 2-16. Combining sorting and paging in a GridView output

Discussion

Unlike the `DataGrid` , the `GridView` provides the ability to perform sorting and paging without having to write any custom code. But because the `GridView` does not provide any indication of the current sort column or order, you may want to write a small amount of custom code to accomplish this, as described in this recipe.

As a first step, the `GridView` , an excellent addition to the ASP.NET 2.0controls, provides the ability to implement standard sorting and paging with little code. Sorting and paging are enabled by setting th `AllowSorting` and `AllowPaging` attributes to `TRue` .

```
<asp:GridView ID="gvBooks" Runat="Server"
    AllowPaging="true"
    AllowSorting="true"
    ShowHeader="true"
  …
```

The ShowHeader attribute must be set to true to support sorting since the header provides the controls used to perform the sorting.

In addition, the HeaderText for each column that needs to support sorting cannot be set to an empty string. Otherwise, the control used to perform the sort operation for the column will not be rendered.

The Page_Load event handler in our application initializes the SqlDataSource and then sets the DataSourceID of our GridView to the ID of the SqlDataSource .

The GridView 's built-in sorting and paging works only if the DataSourceID property of the GridView is set to the ID of the data source. If you use the DataSource property instead, you will have to implement all of the same event handlers that are required to perform sorting and paging with a DataGrid (see Recipe 2.11).

By default, the SortExpression property of the GridView is set to an empty string and the SortDirection property is set to Ascending . To set the sort column and sort to meet your needs, you will need to call the Sort method of the GridView passing the desired sort expression and order. This is required since the SortExpression and SortDirection properties are read-only and cannot be set directly. In our application, we set the initial sort column to the Title column and the sort order to Ascending .

```
If (Not Page.IsPostBack) Then
 'perform the initial sort on the first column in ascending order
 gvBooks.Sort(gvBooks.Columns(0).SortExpression, _
    SortDirection.Ascending)
End If


if (!Page.IsPostBack)
{
  // perform the initial sort on the first column in ascending order
  gvBooks.Sort(gvBooks.Columns[0].SortExpression,
   SortDirection.Ascending);
}
```

Call the Sort method only if the page is initially being displayed. If theSort method is called for each postback, the data will always be sorted using the defaults.

To mark the current sort column and order in the GridView' s header, custom code is required in the RowCreated event handler. Since the RowCreated event occurs for each row created in the GridView , you must first check to see if the row that has been created is the header.

VB

```
If (e.Row.RowType = DataControlRowType.Header) Then

  …

End If 'If (gvBooks.SortExpression.Equals(String.Empty))
```

C#

```
if (e.Row.RowType == DataControlRowType.Header)
{
  …

} // if (e.Row.RowType == DataControlRowType.Header)
```

If the row that has been created is the header, you will need to loop through all of the columns in the GridView to determine which column matches the current sort expression. When the current sort column is determined, you need to check the sort directionand then add an HTML image to the controls in the header column to indicate the sort direction.

```
If (col.SortExpression.Equals(gvBooks.SortExpression)) Then
  'this is the sort column so determine whether the ascending or
  'descending image needs to be included
  image = New HtmlImage()
  image.Border = 0
  If (gvBooks.SortDirection = SortDirection.Ascending) Then
    image.Src = "images/sort_ascending.gif"
  Else
    image.Src = "images/sort_descending.gif"
  End If

  'add the image to the column header
  e.Row.Cells(index).Controls.Add(image)


if (col.SortExpression.Equals(gvBooks.SortExpression))
{
  // this is the sort column so determine whether the ascending or
  // descending image needs to be included
  image = new HtmlImage();
  image.Border = 0;
  if (gvBooks.SortDirection == SortDirection.Ascending)
  {
   image.Src = "images/sort_ascending.gif";
  }
  else
  {
   image.Src = "images/sort_descending.gif";
  }

  // add the image to the column header
  e.Row.Cells[index].Controls.Add(image);
 } // if (col.SortExpression.Equals(gvBooks.SortExpression))
```

See Also

Recipes 2.11 and 2.15

Example 2-39. Combining sorting and paging in a GridView (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02GridViewWithSortingAndPagingVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02GridViewWithSortingAndPagingVB"
 Title="GridView With Sorting and Paging" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  GridView With Sorting and Paging (VB)
 </div>
 <asp:SqlDataSource ID="dSource" runat="server" />
 <asp:GridView ID="gvBooks" Runat="Server"
    AllowPaging="true"
    AllowSorting="true"
    ShowHeader="true"
    AutoGenerateColumns="false"
    BorderColor="#000080"
    BorderStyle="Solid"
    BorderWidth="2px"
    Caption=""
    HorizontalAlign="Center"
    Width="90%"
    PageSize="5"
    PagerSettings-Mode="Numeric"
    PagerSettings-PageButtonCount="5"
    PagerSettings-Position="Bottom"
    PagerStyle-HorizontalAlign="Center"
    PagerSettings-NextPageText="Next"
    PagerSettings-PreviousPageText="Prev"
    PagerStyle-CssClass="pagerText"
    OnRowCreated="gvBooks_RowCreated" >
 <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
 <RowStyle cssClass="tableCellNormal" />
 <AlternatingRowStyle cssClass="tableCellAlternating" />
 <Columns>
  <asp:BoundField DataField="Title"
    HeaderText="Title "
    SortExpression="Title" />
  <asp:BoundField DataField="ISBN"
    HeaderText="ISBN "
    ItemStyle-HorizontalAlign="Center"
    SortExpression="ISBN" />
  <asp:BoundField DataField="Publisher"
    HeaderText="Publisher "
    ItemStyle-HorizontalAlign="Center"
    SortExpression="Publisher" />
  </Columns>
 </asp:GridView>
</asp:Content>
```

Example 2-40. Combining sorting and paging in a GridView (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' CH02GridViewWithSortingAndPagingVB.aspx
 ''' </summary>
 Partial Class CH02GridViewWithSortingAndPagingVB
  Inherits System.Web.UI.Page
  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
```

```
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Private Sub Page_Load(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles Me.Load
 'configure the data source to get the data from the database
 'NOTE: This code must be executed anytime the page is rendered
 '  including postbacks
 dSource.ConnectionString = ConfigurationManager. _
  ConnectionStrings("dbConnectionString").ConnectionString
 dSource.DataSourceMode = SqlDataSourceMode.DataSet
 dSource.ProviderName = "System.Data.OleDb"
 dSource.SelectCommand = "SELECT Title, ISBN, Publisher " & _
    "FROM Book " &_
    "ORDER BY Title"

 'set the data source ID for the GridView
 'NOTE: The DataSourceID must be used instead of the DataSource if the
 '  automatic sorting/paging in GridView are to be used.
 gvBooks.DataSourceID = dSource.ID

 If (Not Page.IsPostBack) Then
  'perform the initial sort on the first column in ascending order
  gvBooks.Sort(gvBooks.Columns(0).SortExpression, _
  SortDirection.Ascending)
 End If

End Sub 'Page_Load

'''**************************************************************************
''' <summary>
''' This routine provides the event handler for the GridView's row created

''' event. It is responsible for setting the icon in the header row to
''' indicate the current sort column and sort order
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub gvBooks_RowCreated(ByVal sender As Object, _
  ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs)
 Dim index As Integer
 Dim col As DataControlField = Nothing
 Dim image As HtmlImage = Nothing

 If (e.Row.RowType = DataControlRowType.Header) Then
  'loop through the columns in the gridview updating the header to
  'mark which column is the sort column and the sort order
  For index = 0 To gvBooks.Columns.Count - 1
  col = gvBooks.Columns(index)

  'check to see if this is the sort column
  If (col.SortExpression.Equals(gvBooks.SortExpression)) Then
```

```
    'this is the sort column so determine whether the ascending or
    'descending image needs to be included
    image = New HtmlImage()
    image.Border = 0
    If (gvBooks.SortDirection = SortDirection.Ascending) Then
    image.Src = "images/sort_ascending.gif"
    Else
    image.Src = "images/sort_descending.gif"
    End If

    'add the image to the column header
    e.Row.Cells(index).Controls.Add(image)
    End If 'If (col.SortExpression = sortExpression)
    Next index
   End If 'If (gvBooks.SortExpression.Equals(String.Empty))
  End Sub 'gvBooks_RowCreated
 End Class 'CH02GridViewWithSortingAndPagingVB
End Namespace
```

Example 2-41. Combining sorting and paging in a GridView (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02GridViewWithSortingAndPagingCS.aspx

 /// </summary>
 public partial class CH02GridViewWithSortingAndPagingCS
  : System.Web.UI.Page
{
 ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event. It
 /// is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 private void Page_Load(Object sender,
    System.EventArgs e)
```

```
{
 // configure the data source to get the data from the database
 // NOTE: This code must be executed anytime the page is rendered
 //   including postbacks
 dSource.ConnectionString = ConfigurationManager.
  ConnectionStrings["dbConnectionString"].ConnectionString;
 dSource.DataSourceMode = SqlDataSourceMode.DataSet;
 dSource.ProviderName = "System.Data.OleDb";
 dSource.SelectCommand = "SELECT Title, ISBN, Publisher " +
    "FROM Book " +
    "ORDER BY Title";

 // set the data source ID for the GridView
 // NOTE: The DataSourceID must be used instead of the DataSource if the
 //  automatic sorting/paging in GridView are to be used.
 gvBooks.DataSourceID = dSource.ID;

 if (!Page.IsPostBack)
 {
  // perform the initial sort on the first column in ascending order
  gvBooks.Sort(gvBooks.Columns[0].SortExpression,
   SortDirection.Ascending);
 }
} // Page_Load

///*************************************************************************
/// <summary>
/// This routine provides the event handler for the GridView's row created
/// event. It is responsible for setting the icon in the header row to
/// indicate the current sort column and sort order
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void gvBooks_RowCreated(Object sender,
    System.Web.UI.WebControls.GridViewRowEventArgs e)
{
DataControlField col = null;
  HtmlImage image = null;

  if (e.Row.RowType == DataControlRowType.Header)
  {
   // loop through the columns in the gridview updating the header to
   // mark which column is the sort column and the sort order
   for (int index = 0; index < gvBooks.Columns.Count; index++)
   {
   col = gvBooks.Columns[index];

   // check to see if this is the sort column
   if (col.SortExpression.Equals(gvBooks.SortExpression))
   {
   // this is the sort column so determine whether the ascending or
```

```
    // descending image needs to be included
    image = new HtmlImage();
    image.Border = 0;
    if (gvBooks.SortDirection == SortDirection.Ascending)
    {
    image.Src = "images/sort_ascending.gif";
    }
    else
    {
    image.Src = "images/sort_descending.gif";
    }

    // add the image to the column header
    e.Row.Cells[index].Controls.Add(image);
    } // if (col.SortExpression.Equals(gvBooks.SortExpression))
    } // for index
   } // if (e.Row.RowType == DataControlRowType.Header)
  } //gvBooks_RowCreated
 } // CH02GridViewWithSortingAndPagingCS
}
```

PREV

Recipe 2.16. Updating a GridView Without Refreshing the Whole Page

Problem

You want to use a GridView and have its contents be updated without refreshing the whole page when the user performs a sort or paging operation.

Solution

Implement the solution described in Recipe 2.14 and then set the EnableSortingAndPagingCallbacks attribute of the GridView to TRue.

Discussion

The GridView control provides built-in support for making callbacks to the server when a new sort order or page of data is requested by the user, and, in the process, retrieves the most recent data and updates the contents of the GridView, all without having to refresh the whole page. All you have to do to implement this capability is to set the EnableSortingAndPagingCallbacks attribute of the GridView to TRue. ASP.NET handles everything else for you.

```
<asp:GridView ID="gvBooks" Runat="Server"
   AllowPaging="true"
   AllowSorting="true"
   AutoGenerateColumns="false"
   BorderColor="#000080"
   BorderStyle="Solid"
   BorderWidth="2px"
   Caption=""
   HorizontalAlign="Center"
   Width="90%"
   PageSize="5"
   PagerSettings-Mode="Numeric"
   PagerSettings-PageButtonCount="5"
   PagerSettings-Position="Bottom"
   PagerStyle-HorizontalAlign="Center"
   PagerSettings-NextPageText="Next"
   PagerSettings-PreviousPageText="Prev"
   PagerStyle-CssClass="pagerText"
   OnRowCreated="gvBooks_RowCreated"
```

```
    EnableSortingAndPagingCallbacks="true" >
```

ASP.NET handles the partial page refresh for you by outputting client-side JavaScript to perform an asynchronous callback to the server and handle the callback when the updated data is returned from the server. We won't delve into the `JavaScipt` here, but you can see it yourself by implementing the `EnableSortingAndPagingCallbacks` attribute in your application and then using the View Source command in the browser.

> The ability to update the contents of a `GridView` without refreshing the whole page is not supported in all browsers. Only Internet Explorer 5.5 (and later), Netscape 6.0 (and later), and Firefox are supported.

See Also

Recipe 2.14

Recipe 2.17. Editing Data in a GridView

Problem

You want to allow the user to edit the data within the table displayed by a `GridView` .

Solution

Add a `GridView` and an updateable data source, such as an `asp:SqlDataSource` , to the *.aspx* file, set the `AutoGenerateEditButton` attribute of the `GridView` control to true, add `EditItemTemplate` elements for each column that is to be editable, and initialize the properties of the data source in the code-behind.

In the *.aspx* file:

- Add a `GridView` control where the data is to be displayed.
- Add an `asp:SqlDataSource` .
- Set the `AutoGenerateEditButton` attribute of the `GridView` to true.
- Add `EditItemTemplate` elements for each editable column.

In the `Page_Init` event handler of the code-behind class for the page, use the .NET language of your choice to:

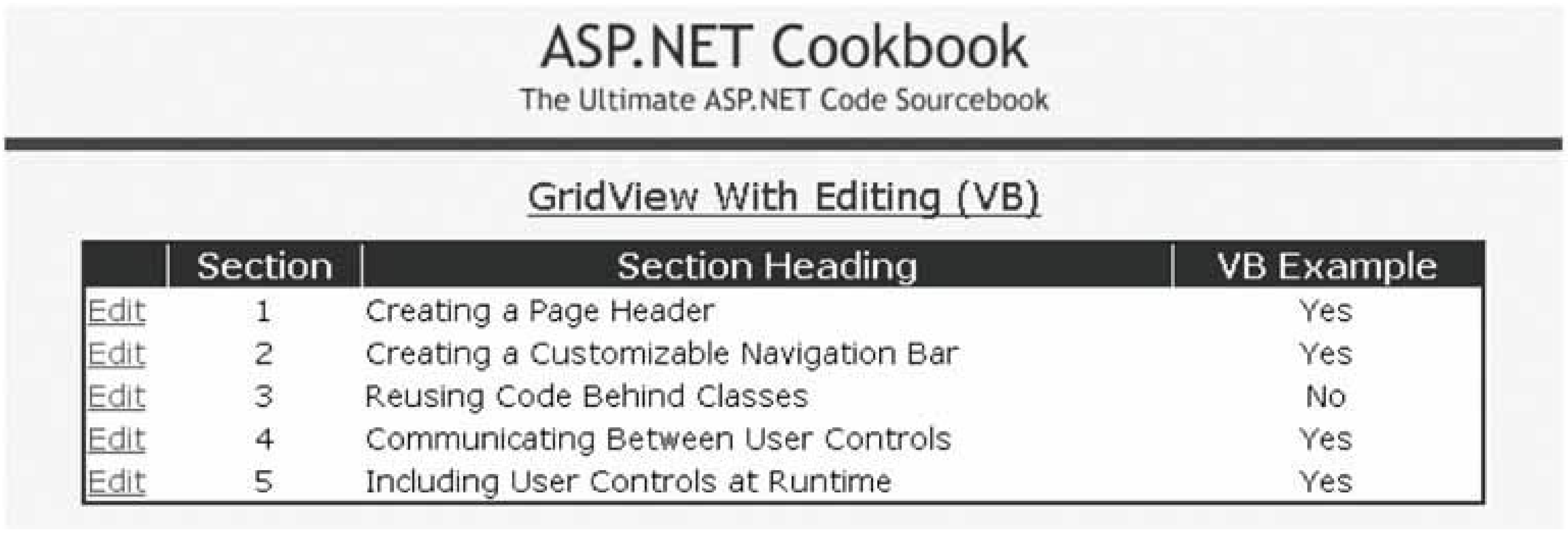
- Initialize the `SelectCommand` property of the `SqlDataSource` to the SQL statement used to get the data to display from the database.
- Initialize the `UpdateCommand` property of the `SqlDataSource` to the SQL statement used to update with parameters for the data the user can edit.
- Add the parameters for the data the user can update to the `UpdateParameters` collection of the `SqlDataSource` .
- Set the `DataKeyNames` collection of the `SqlDataSource` to the primary key(s) used to identify a unique row of data in the `GridView` .

Figure 2-17 shows the output of a typical example in normal mode and Figure 2-18 shows the output in edit mode. Examples 2-42, 2-43 through 2-44 show the *.aspx* file and the code-behind files for an application that produces this output.

Discussion

The GridView reduces the amount of custom code required for editing tabular data over what was required in ASP.NET 1.x (see Recipe 2.13). By binding aGridView to an updateable data source such as the SqlDataSource , the GridView 's built-in editing capability can be used with little custom code.

Figure 2-17. GridView with editingnormal mode


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

GridView With Editing (VB)

	Section	Section Heading	VB Example
Edit	1	Creating a Page Header	Yes
Edit	2	Creating a Customizable Navigation Bar	Yes
Edit	3	Reusing Code Behind Classes	No
Edit	4	Communicating Between User Controls	Yes
Edit	5	Including User Controls at Runtime	Yes

Figure 2-18. GridView with editingrow edit mode

To add an edit button to each row in the GridView automatically, as shown in Figure 2-17, you need to set the AutoGenerateEditButton attribute to true . This will automatically generate the update and cancel buttons when the GridView is in edit mode (see Figure 2-18).

```
<asp:GridView ID="gvProblems" runat="server"
    BorderColor="#000080"
    BorderWidth="2px"
   AutoGenerateColumns="False"
   HorizontalAlign="center"
```

```
 Width="90%"
AutoGenerateEditButton="true" >
```

If you use `asp:BoundField` elements to define the data displayed in the `GridView` , the `GridView` will automatically handle rendering textboxes for editing the data; however, you are limited to editing simple data. If you want to provide the ability for the user to select from a dropdown or provide any other special editing features, you will need to define the displayed data using an `asp:TemplateField` element for each editable column. The `asp:TemplateField` elements need to include an `ItemTemplate` element that defines how the data is to be displayed in normal mode and an `EditItemTemplate` element to define how the data is displayed in edit mode. The `EditItemTemplates` can include dropdowns, radio buttons, or any other control that your application needs.

```
<Columns>
 <asp:BoundField DataField="SectionNumber"
   ItemStyle-HorizontalAlign="Center"
   HeaderText="Section"
   ReadOnly="True" />
 <asp:TemplateField HeaderText="Section Heading"
    ItemStyle-HorizontalAlign="Left">
  <ItemTemplate>
   <%#Eval("SectionHeading")%>
  </ItemTemplate>
  <EditItemTemplate>
   <asp:TextBox id="txtEditName" runat="server"
    Columns="40"
    Text='<%# Bind("SectionHeading") %>' />
   </EditItemTemplate>
  </asp:TemplateField>

 <asp:TemplateField HeaderText="VB Example"
   ItemStyle-HorizontalAlign="Center">
  <ItemTemplate>
   <%#yesNoSelections.Item(CInt(Eval("HasVBExample")))%>
  </ItemTemplate>
  <EditItemTemplate>
   <asp:DropDownList id="selHasVBSample" runat="server"
   DataSource="<%# yesNoSelections %>"
   DataTextField="Text"
   DataValueField="Value"
   SelectedValue='<%# Bind("HasVBExample") %>' />
  </EditItemTemplate>
 </asp:TemplateField>
</Columns>
```

In our example, we have two editable columns. The first column contains simple text. To provide more control over the size of the textbox displayed in edit mode, we have used an `asp:TextBox` and set the number of columns to `40` .

The second column contains a yes/no value. We are using an asp:DropDownList control when the GridView is in edit mode to limit the choices the user can enter. In addition, we are binding the asp:DropDownList to an ArrayList containing the valid values (described later).

When using the GridView 's built-in editing functionality with EditItemTemplates , the data binding must use the new Bind method instead of the Eval method. When using Bind and the form is posted back to the server, a name/value collection of the edited data is passed to the server and is used by ASP.NET to provide the parameter values for the update command.

The names of the parameters defined in the UpdateCommand and the UpdateParameters collection must match the names of the columns in the data source bound to the controls using the Bind method. If they do not match, the update will fail, indicating the missing parameter value must be specified. If you need the column names in the data source and parameter names to be different, you have to provide custom code to handle the differences.

In the Page_Init event handler in the code-behind, the ArrayList used to provide the valid selections in the asp:DropDownList displayed in edit modeis initialized. In our example, we created the list programmatically. In your application, the list can be created from a database or any other data source, as required.

The SqlDataSource is initialized in the normal manner with a couple of additions. The UpdateCommand property is set to the SQL required to update the data in the database. The SQL needs to use parameters for the values to update as well as the value of the primary key used to identify the unique record being updated.

```
dSource.UpdateCommand = "UPDATE EditProblem " & _
   "SET SectionHeading=@SectionHeading" & _
   ",HasVBExample=@HasVBExample" & _
   " WHERE EditProblemID=@_EditProblemID"

dSource.UpdateCommand = "UPDATE EditProblem " +
   "SET SectionHeading=@SectionHeading" +
   ",HasCSExample=@HasCSExample" +
   " WHERE EditProblemID=@_EditProblemID";
```

In addition, Parameter objects need to be created and added to the UpdateParameters collection of SqlDataSource for each of the data values being changed. A Parameter object does not need to be added for the primary key value. ASP.NET will automatically handle the primary key value(s) from DataKeyNames collection (described below). The parameter names must match the names of the parameters in the SQL update statement.

VB

```
param = New Parameter("SectionHeading", _
     TypeCode.String)
dSource.UpdateParameters.Add(param)
param = New Parameter("HasVBExample", _
     TypeCode.Int32)
dSource.UpdateParameters.Add(param)
```

C#

```
param = new Parameter("SectionHeading",
     TypeCode.String);
dSource.UpdateParameters.Add(param);
param = new Parameter("HasCSExample",
     TypeCode.Int32);
dSource.UpdateParameters.Add(param);
```

The `DataKeyNames` collection is set to a string array containing the names of the column(s) used in the database as the primary key. This can be one or more columns.

```
Dim dataKeys(0) As String

 …

dataKeys(0) = "EditProblemID"
gvProblems.DataKeyNames = dataKeys
```

```
String[] dataKeys;

 …

dataKeys = new string[1] {"EditProblemID"};
gvProblems.DataKeyNames = dataKeys;
```

Finally, the `DataSourceID` property is set to the ID of the `SqlDataSource` :

```
gvProblems.DataSourceID = dSource.ID
```

```
gvProblems.DataSourceID = dSource.ID;
```

The GridView 's built-in editing will work only if the DataSourceID property of the GridView is set to the ID of the data source. If the DataSource property is used instead, you will have to implement all of the same event handlers required to perform editing with a DataGrid (see Recipe 2.13).

By using a GridView with an updateable data source and following the coding conventions for column and parameter names described in this recipe, editing can be implemented with a minimal amount of custom code.

The data source must be initialized in the Page_Init event handler. If the data source is initialized later in the page life cycle, ASP.NET will not have the information it needs to use the built-in editing functionality.

See Also

Recipe 2.13

Example 2-42. GridView with editing (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02GridViewWithEditingVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02GridViewWithEditingVB"
 Title="GridView With Editing" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  GridView With Editing (VB)
 </div>
 <asp:SqlDataSource id="dSource" runat="server" />
 <asp:GridView ID="gvProblems" runat="server"

     BorderColor="#000080"
     BorderWidth="2px"
     AutoGenerateColumns="False"
     HorizontalAlign="center"
     Width="90%"
     AutoGenerateEditButton="true" >
  <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader"
  <RowStyle cssClass="tableCellNormal" />
  <AlternatingRowStyle cssClass="tableCellAlternating" />
  <Columns>
   <asp:BoundField DataField="SectionNumber"
    ItemStyle-HorizontalAlign="Center"
    HeaderText="Section"
    ReadOnly="True" />
   <asp:TemplateField HeaderText="Section Heading"
     ItemStyle-HorizontalAlign="Left">
    <ItemTemplate>
    <%#Eval("SectionHeading")%>
    </ItemTemplate>
    <EditItemTemplate>
    <asp:TextBox id="txtSectionHeading" runat="server"
     Columns="40"
     Text='<%# Bind("SectionHeading") %>' />
    </EditItemTemplate>
   </asp:TemplateField>

   <asp:TemplateField HeaderText="VB Example"
     ItemStyle-HorizontalAlign="Center">
    <ItemTemplate>
    <%#yesNoSelections.Item(CInt(Eval("HasVBExample")))%>
    </ItemTemplate>
    <EditItemTemplate>
    <asp:DropDownList id="selHasVBSample" runat="server"
    DataSource="<%# yesNoSelections %>"
    DataTextField="Text"
    DataValueField="Value"
    SelectedValue='<%# Bind("HasVBExample") %>' />
    </EditItemTemplate>
   </asp:TemplateField>
  </Columns>
 </asp:GridView>
</asp:Content>
```

Example 2-43. GridView with editing (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' CH02GridViewWithEditingVB.aspx
 ''' </summary>
 Partial Class CH02GridViewWithEditingVB
  Inherits System.Web.UI.Page

  'The following variable contains the list of yes/no selections used in
  'the dropdown lists and is declared protected to provide access to the
  'data from the aspx page
  Protected yesNoSelections As ArrayList

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page init event. It
  ''' is responsible for initializing the data source and the grid view
  ''' on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Init(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Init
   Dim dataKeys(0) As String
   Dim param As Parameter

   'build the arraylist with the acceptable responses to the
   '"Has VB Sample" field
   yesNoSelections = New ArrayList(2)
   yesNoSelections.Add(New ListItem("No", "0"))
   yesNoSelections.Add(New ListItem("Yes", "1"))

   'configure the data source to get the data from the database
   'NOTE: This code must be executed anytime the page is rendered
   '  including postbacks
   dSource.ConnectionString = ConfigurationManager. _
    ConnectionStrings("sqlConnectionString").ConnectionString
   dSource.DataSourceMode = SqlDataSourceMode.DataSet
   dSource.ProviderName = "System.Data.SqlClient"
   dSource.SelectCommand = "SELECT EditProblemID, SectionNumber" & _
      ", SectionHeading, HasVBExample" & _
      " FROM EditProblem" & _
      " ORDER BY SectionNumber"
   dSource.UpdateCommand = "UPDATE EditProblem " & _
      "SET SectionHeading=@SectionHeading" & _
      ",HasVBExample=@HasVBExample" & _
      " WHERE EditProblemID=@_EditProblemID"
   param = New Parameter("SectionHeading", _
     TypeCode.String)
   dSource.UpdateParameters.Add(param)

   param = New Parameter("HasVBExample", _
     TypeCode.Int32)
   dSource.UpdateParameters.Add(param)

   'set the source of the data and the data keys for the gridview control
   dataKeys(0) = "EditProblemID"
   gvProblems.DataKeyNames = dataKeys
   gvProblems.DataSourceID = dSource.ID
  End Sub 'Page_Init
End Class 'CH02GridViewWithEditingVB
End Namespace
```

Example 2-44. GridView with editing (.cs)

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02GridViewWithEditingCS.aspx
 /// </summary>
 public partial class CH02GridViewWithEditingCS : System.Web.UI.Page
 {
  // The following variable contains the list of yes/no selections used in
  // the dropdown lists and is declared protected to provide access to the
  // data from the aspx page
  protected ArrayList yesNoSelections;

  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page init event. It
  /// is responsible for initializing the data source and the grid view
  /// on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Init(object sender,
        EventArgs e)
  {
   String[] dataKeys;
   Parameter param;

   // build the arraylist with the acceptable responses to the
   // "Has C# Sample" field
   yesNoSelections = new ArrayList(2);
   yesNoSelections.Add(new ListItem("No", "0"));
   yesNoSelections.Add(new ListItem("Yes", "1"));

   // configure the data source to get the data from the database
   // NOTE: This code must be executed anytime the page is rendered
   //   including postbacks
   dSource.ConnectionString = ConfigurationManager.
```

```
     ConnectionStrings["sqlConnectionString"].ConnectionString;
    dSource.DataSourceMode = SqlDataSourceMode.DataSet;
    dSource.ProviderName = "System.Data.SqlClient";
    dSource.SelectCommand = "SELECT EditProblemID, SectionNumber" +
       ", SectionHeading, HasCSExample" +
       " FROM EditProblem" +
       " ORDER BY SectionNumber";
    dSource.UpdateCommand = "UPDATE EditProblem " +
       "SET SectionHeading=@SectionHeading" +
       ",HasCSExample=@HasCSExample" +
       " WHERE EditProblemID=@_EditProblemID";
    param = new Parameter("SectionHeading",
      TypeCode.String);
    dSource.UpdateParameters.Add(param);
    param = new Parameter("HasCSExample",
      TypeCode.Int32);
    dSource.UpdateParameters.Add(param);

    // set the source of the data and the data keys for the gridview control
    dataKeys = new string[1] {"EditProblemID"};
    gvProblems.DataKeyNames = dataKeys;
    gvProblems.DataSourceID = dSource.ID;
  } // Page_Init
 } // CH02GridViewWithEditingCS
}
```

Recipe 2.18. Inserting a Row Within a GridView

Problem

You want to provide the ability for a user to insert a new row of data within a`GridView` .

Solution

Add a `GridView` and an updateable data source, such as an `asp:SqlDataSource` , to the *.aspx* file, set the `ShowFooter` attribute of the `GridView` control to `TRue` , add `Footer-Template` elements for each column that is to be inserted, add an Insert button in the footer, and initialize the properties of the data source in the code-behind. When the user clicks the Insert button, set the parameter values from the entered data and use the data source to insert the data in the`GridView's RowCommand` event handler in the code-behind.

In the .aspx file:

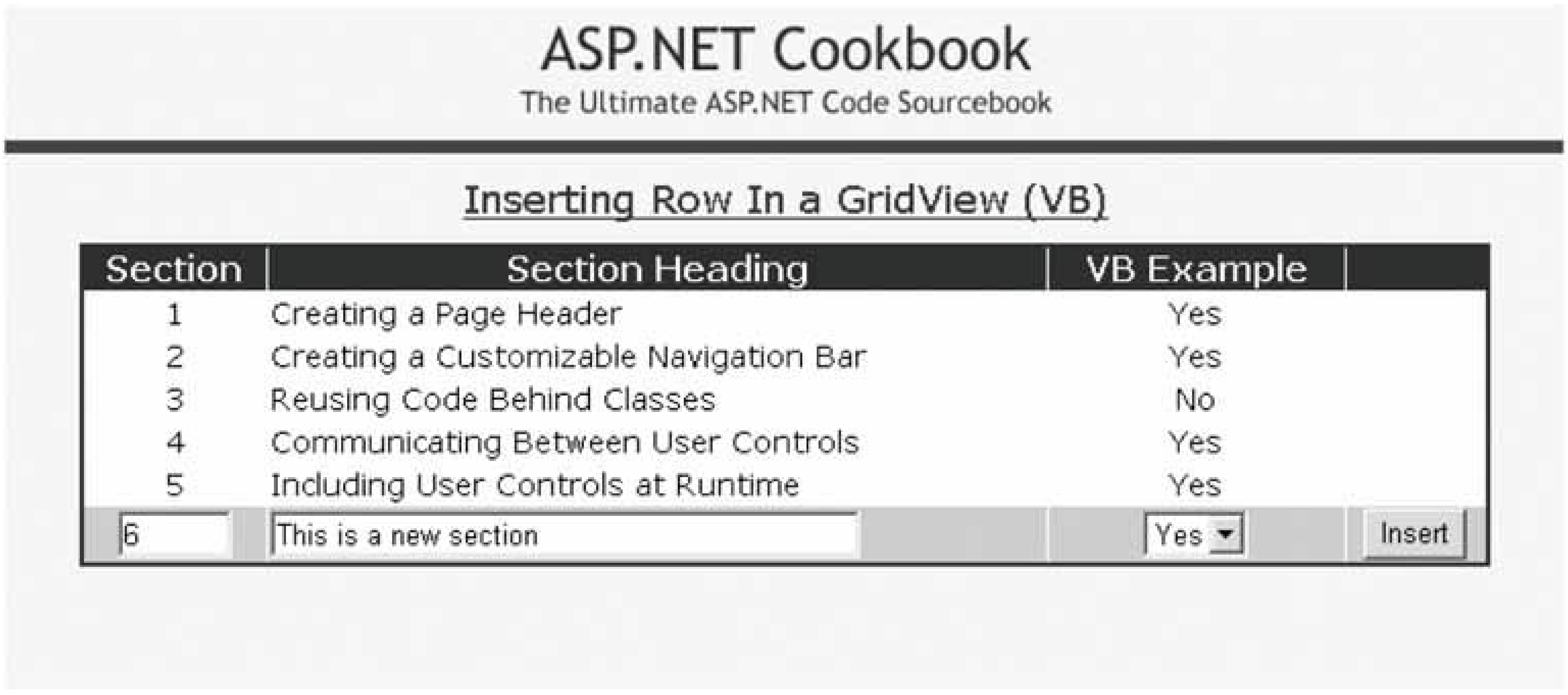
1. Add a `GridView` control where the data is to be displayed.
2. Add an `asp:SqlDataSource` .
3. Set the `ShowFooter` attribute of the `GridView` to `TRue` .
4. Add `FooterTemplate` elements for each column that is to be inserted.

In the `Page_Init` event handler of the code-behind class for the page, use the .NET language of your choice to:

1. Initialize the `SelectCommand` property of the `SqlDataSource` to the SQL statement used to get the data to display from the database.
2. Initialize the `InsertCommand` property of the `SqlDataSource` to the SQL statement (with parameters) used to insert in the database the data the user enters.
3. Add the parameters for the data the user can update to the`InsertParameters` collection of the `SqlDataSource` .
4. In the `RowCommand` event handler, set the values of the `InsertParameters` of the `SqlDataSource` from the controls containing the data entered by the user and then call the`Insert` method of the `SqlDataSource` .

Figure 2-19 shows the output of a typical example in normal mode and Figure 2-18shows the output in our application. Examples 2-45 , 2-46 through 2-47 show the *.aspx* file and the code-behind files for an application that produces this output.

Figure 2-19. GridView with row insert output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Inserting Row In a GridView (VB)

Section	Section Heading	VB Example	
1	Creating a Page Header	Yes	
2	Creating a Customizable Navigation Bar	Yes	
3	Reusing Code Behind Classes	No	
4	Communicating Between User Controls	Yes	
5	Including User Controls at Runtime	Yes	
6	This is a new section	Yes	Insert

Discussion

Most applications provide the ability to add new records to a tabular display of data. Though the GridView does not directly support inserting a new record using an updateable data source, such as the SqlDataSource , it can be altered to provide the ability to insert a new record when combined witl a small amount of custom code.

In our example, we use the footer row of the GridView to provide a location for the textboxes and drop-down list necessary to enter the data for a new row. In addition, we add a new column on the right side of the GridView to provide a location for an Insert button, as shown in Figure 2-19. To make the footer visible, the ShowFooter attribute must be set to true .

```
<asp:GridView ID="gvProblems" runat="server"
    BorderColor="#000080"
    BorderWidth="2px"
    AutoGenerateColumns="False"
    HorizontalAlign="center"
    Width="90%"
    ShowFooter="true"
    OnRowCommand="gvProblems_RowCommand">
```

A FooterTemplate element is provided for each asp:TemplateField element to define the HTML that is to be rendered in the footer. The first and second columns contain an asp:TextBox control, the third column contains an asp:DropDownList control, and the fourth column contains an asp:Button control.

```
<Columns>
 <asp:TemplateField HeaderText="Section"
   ItemStyle-HorizontalAlign="Center"
   FooterStyle-HorizontalAlign="Center">
  <ItemTemplate>
   <%#Eval("SectionNumber")%>
  </ItemTemplate>
  <FooterTemplate>
   <asp:TextBox id="txtSectionNumber" runat="server"
    Columns="3" />
  </FooterTemplate>
 </asp:TemplateField>

 <asp:TemplateField HeaderText="Section Heading"
   ItemStyle-HorizontalAlign="Left"
   FooterStyle-HorizontalAlign="Left">
  <ItemTemplate>
   <%#Eval("SectionHeading")%>
  </ItemTemplate>
  <FooterTemplate>
   <asp:TextBox id="txtSectionHeading" runat="server"
    Columns="40" />
  </FooterTemplate>
 </asp:TemplateField>

 <asp:TemplateField HeaderText="VB Example"
   ItemStyle-HorizontalAlign="Center"
   FooterStyle-HorizontalAlign="Center">
  <ItemTemplate>
   <%#yesNoSelections.Item(CInt(Eval("HasVBExample")))%<

  </ItemTemplate>
  <FooterTemplate>
   <asp:DropDownList id="selHasVBSample" runat="server"
   DataSource="<%# yesNoSelections %>"
   DataTextField="Text"
   DataValueField="Value" />
  </FooterTemplate>
 </asp:TemplateField>

 <asp:TemplateField FooterStyle-HorizontalAlign="Center">
  <FooterTemplate>
   <asp:Button ID="btnInsert" runat="server"
    Text="Insert"
    CommandName="Insert" />
  </FooterTemplate>
 </asp:TemplateField>
</Columns>
```

In the Page_Init event handler in the code-behind, the ArrayList used to provide the valid selections in the asp:DropDownList is initialized and bound to the drop-down list. In our example, we programmatically created the list. In your application, the list can be created from a database or any other data source, as required.

The SqlDataSource is initialized in the normal manner with a couple of additions. The InsertCommand property is set to the SQL required to insert the data in the database. The SQL needs to use parameters for the data values.

VB

```
dSource.InsertCommand = "INSERT INTO EditProblem " & _
   " (SectionNumber, SectionHeading, HasVBExample)" & _
   " VALUES" & _
   " (@SectionNumber,@SectionHeading,@HasVBExample)"
```

C#

```
dSource.InsertCommand = "INSERT INTO EditProblem " +
   " (SectionNumber, SectionHeading, HasCSExample)" +
   " VALUES" +
   " (@SectionNumber,@SectionHeading,@HasCSExample)";
```

In addition, Parameter objects need to be created and added to the InsertParameters collection of SqlDataSource for each of the data values in the inserted row:

```
param = New Parameter("SectionNumber", _
    TypeCode.Int32)
dSource.InsertParameters.Add(param)
param = New Parameter("SectionHeading", _
   TypeCode.String)
dSource.InsertParameters.Add(param)
param = New Parameter("HasVBExample", _
    TypeCode.Int32)
dSource.InsertParameters.Add(param)
```

```
param = new Parameter("SectionNumber",
    TypeCode.Int32);
dSource.InsertParameters.Add(param);
param = new Parameter("SectionHeading",
   TypeCode.String);


dSource.InsertParameters.Add(param);
param = new Parameter("HasCSExample",
    TypeCode.Int32);
dSource.InsertParameters.Add(param);
```

Finally, the DataSourceID property needs to be set to the ID of the SqlDataSource :

VB

```
gvProblems.DataSourceID = dSource.ID
```

C#

```
gvProblems.DataSourceID = dSource.ID;
```

In the RowCommand event handler (gvProblems_RowCommand in our example), the data entered by the user is retrieved from the controls in the footer and used to set the values for theInsertParameters :

VB

```
get the section number and set the parameter value
tBox = CType(gvProblems.FooterRow.FindControl("txtSectionNumber"), _
   TextBox)
dSource.InsertParameters("SectionNumber").DefaultValue = tBox.Text

'get the section heading and set the parameter value
tBox = CType(gvProblems.FooterRow.FindControl("txtSectionHeading"), _
   TextBox)
dSource.InsertParameters("SectionHeading").DefaultValue = tBox.Text

'get the has sample selection and set the parameter value
ddList = CType(gvProblems.FooterRow.FindControl("selHasVBSample"), _
   DropDownList)
dSource.InsertParameters("HasVBExample").DefaultValue = _
          ddList.SelectedItem.Value

// get the section number and set the parameter value
tBox = (TextBox)(gvProblems.FooterRow.FindControl("txtSectionNumber"));
dSource.InsertParameters["SectionNumber"].DefaultValue = tBox.Text;

// get the section heading and set the parameter value
tBox = (TextBox)(gvProblems.FooterRow.FindControl("txtSectionHeading"));
dSource.InsertParameters["SectionHeading"].DefaultValue = tBox.Text;

// get the has sample selection and set the parameter value
ddList = (DropDownList)(gvProblems.FooterRow.FindControl("selHasCSSample"));
dSource.InsertParameters["HasCSExample"].DefaultValue =
          ddList.SelectedItem.Value;
```

Once the parameter values for the insert command have been set, theInsert method of the data source is called to perform the insert operation:

VB
```
dSource.Insert()
```

C#
```
dSource.Insert();
```

Updateable data sources, like the SqlDataSource used in this example, significantly reduce the amount of custom code required to implement common functionality, such as inserting a new record. This same approach can be used with a DataGrid , provided your application requires the use of a DataGrid .

Example 2-45. GridView with row insert (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02GridViewWithInsertVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02GridViewWithInsertVB"
 Title="Inserting Row In a GridView" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Inserting Row In a GridView (VB)
 </div>
 <asp:SqlDataSource id="dSource" runat="server" />
 <asp:GridView ID="gvProblems" runat="server"
    BorderColor="#000080"
    BorderWidth="2px"
    AutoGenerateColumns="False"
    HorizontalAlign="center"
    Width="90%"
    ShowFooter="true"
    OnRowCommand="gvProblems_RowCommand">
  <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
  <RowStyle cssClass="tableCellNormal" />
  <AlternatingRowStyle cssClass="tableCellAlternating" />
  <FooterStyle CssClass="tableCellSelected" />
  <Columns>
   <asp:TemplateField HeaderText="Section"
     ItemStyle-HorizontalAlign="Center"
     FooterStyle-HorizontalAlign="Center">
    <ItemTemplate>
    <%#Eval("SectionNumber")%>
    </ItemTemplate>
    <FooterTemplate>
    <asp:TextBox id="txtSectionNumber" runat="server"
    Columns="3" />
    </FooterTemplate>
   </asp:TemplateField>
```

```
    <asp:TemplateField HeaderText="Section Heading"
      ItemStyle-HorizontalAlign="Left"
      FooterStyle-HorizontalAlign="Left">
     <ItemTemplate>
     <%#Eval("SectionHeading")%>
     </ItemTemplate>
     <FooterTemplate>
     <asp:TextBox id="txtSectionHeading" runat="server"
     Columns="40" />
     </FooterTemplate>
    </asp:TemplateField>

    <asp:TemplateField HeaderText="VB Example"
      ItemStyle-HorizontalAlign="Center"
      FooterStyle-HorizontalAlign="Center">
    <ItemTemplate>
     <%#yesNoSelections.Item(CInt(Eval("HasVBExample")))%>

     </ItemTemplate>
     <FooterTemplate>
     <asp:DropDownList id="selHasVBSample" runat="server"
     DataSource="<%# yesNoSelections %>"
     DataTextField="Text"
     DataValueField="Value" />
     </FooterTemplate>>
    </asp:TemplateField>

    <asp:TemplateField FooterStyle-HorizontalAlign="Center">
     <FooterTemplate>
     <asp:Button ID="btnInsert" runat="server"
       Text="Insert"
       CommandName="Insert" />
     </FooterTemplate>
    </asp:TemplateField>
   </Columns>
  </asp:GridView>
</asp:Content>
```

Example 2-46. GridView with row insert (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data

Namespace ASPNetCookbook.VBExamples
```

```
''' <summary>
''' This class provides the code behind for
'''  CH02GridViewWithInsertVB.aspx
''' </summary>
Partial  Class  CH02GridViewWithInsertVB
 Inherits System.Web.UI.Page

 'The following variable contains the list of yes/no selections used in
 'the dropdown lists and is declared protected to provide access to the
 'data from the aspx page
 Protected yesNoSelections As ArrayList

 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page init event. It
 ''' is responsible for initializing the data source and the grid view
 ''' on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub Page_Init(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Init
  Dim param As Parameter

   'build the arraylist with the acceptable responses to the
   '"Has VB Sample" field
   yesNoSelections = New ArrayList(2)
   yesNoSelections.Add(New ListItem("No", "0"))
   yesNoSelections.Add(New ListItem("Yes", "1"))

   'configure the data source to get the data from the database
   'NOTE: This code must be executed anytime the page is rendered
   '  including postbacks
   dSource.ConnectionString = ConfigurationManager. _
    ConnectionStrings("sqlConnectionString").ConnectionString
   dSource.DataSourceMode = SqlDataSourceMode.DataSet
   dSource.ProviderName = "System.Data.SqlClient"
   dSource.SelectCommand = "SELECT EditProblemID, SectionNumber" & _
      ", SectionHeading, HasVBExample" & _
      " FROM EditProblem" & _
      " ORDER BY SectionNumber"

   dSource.InsertCommand = "INSERT INTO EditProblem " & _
      " (SectionNumber, SectionHeading, HasVBExample)" & _
      " VALUES" & _
      " (@SectionNumber,@SectionHeading,@HasVBExample)"
   param = New Parameter("SectionNumber", _
     TypeCode.Int32)
   dSource.InsertParameters.Add(param)
   param = New Parameter("SectionHeading", _
     TypeCode.String)
    dSource.InsertParameters.Add(param)
    param = New Parameter("HasVBExample", _
      TypeCode.Int32)
    dSource.InsertParameters.Add(param)

    'set the source of the data for the gridview control
    gvProblems.DataSourceID = dSource.ID
  End Sub 'Page_Init

  '''**************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the GridView row command.
  ''' It is responsible for processing the Add button click.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub gvProblems_RowCommand(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.GridViewCommandEventArgs)
    Dim tBox As TextBox
    Dim ddList As DropDownList

    'check to see if the command is for the Insert button
    If (e.CommandName.Equals("Insert")) Then
      'get the section number and set the parameter value

      tBox = CType(gvProblems.FooterRow.FindControl("txtSectionNumber"), _
      TextBox)
      dSource.InsertParameters("SectionNumber").DefaultValue = tBox.Text

      'get the section heading and set the parameter value
      tBox = CType(gvProblems.FooterRow.FindControl("txtSectionHeading"), _
         TextBox)
      dSource.InsertParameters("SectionHeading").DefaultValue = tBox.Text

      'get the has sample selection and set the parameter value
      ddList = CType(gvProblems.FooterRow.FindControl("selHasVBSample"), _
       DropDownList)
      dSource.InsertParameters("HasVBExample").DefaultValue = _
              ddList.SelectedItem.Value

      'insert the row in the database
      dSource.Insert()
    End If
  End Sub 'gvProblems_RowCommand
 End Class 'CH02GridViewWithInsertVB
End Namespace
```

Example 2-47. GridView with row insert (.cs)

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02GridViewWithInsertCS.aspx
 /// </summary>
 public partial class CH02GridViewWithInsertCS : System.Web.UI.Page
 {
  // The following variable contains the list of yes/no selections used in
  // the dropdown lists and is declared protected to provide access to the
  // data from the aspx page
  protected ArrayList yesNoSelections;

  ///***************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page init event. It
  /// is responsible for initializing the data source and the grid view
  /// on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Init(object sender,
        EventArgs e)

  {
   Parameter param;

   // build the arraylist with the acceptable responses to the
   // "Has C# Sample" field
   yesNoSelections = new ArrayList(2);
   yesNoSelections.Add(new ListItem("No", "0"));
   yesNoSelections.Add(new ListItem("Yes", "1"));

   // configure the data source to get the data from the database
   // NOTE: This code must be executed anytime the page is rendered
   //  including postbacks
   dSource.ConnectionString = ConfigurationManager.
    ConnectionStrings["sqlConnectionString"].ConnectionString;
   dSource.DataSourceMode = SqlDataSourceMode.DataSet;
   dSource.ProviderName = "System.Data.SqlClient";
   dSource.SelectCommand = "SELECT EditProblemID, SectionNumber" +
      ", SectionHeading, HasCSExample" +
      " FROM EditProblem" +
      " ORDER BY SectionNumber";
   dSource.InsertCommand = "INSERT INTO EditProblem " +
      " (SectionNumber, SectionHeading, HasCSExample)" +
      " VALUES" +
      " (@SectionNumber,@SectionHeading,@HasCSExample)";
   param = new Parameter("SectionNumber",
     TypeCode.Int32);
   dSource.InsertParameters.Add(param);
   param = new Parameter("SectionHeading",
     TypeCode.String);
   dSource.InsertParameters.Add(param);
   param = new Parameter("HasCSExample",
     TypeCode.Int32);
   dSource.InsertParameters.Add(param);

   // set the source of the data for the gridview control
   gvProblems.DataSourceID = dSource.ID;
  } // Page_Init

  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the GridView row command.
  /// It is responsible for processing the Add button click.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void gvProblems_RowCommand(Object sender,
    System.Web.UI.WebControls.GridViewCommandEventArgs e)
  {
   TextBox tBox;
   DropDownList ddList;

   // check to see if the command is for the Insert button
   if (e.CommandName.Equals("Insert"))
   {

    // get the section number and set the parameter value
    tBox = (TextBox)(gvProblems.FooterRow.FindControl("txtSectionNumber"));
    dSource.InsertParameters["SectionNumber"].DefaultValue = tBox.Text;

    // get the section heading and set the parameter value
    tBox = (TextBox)(gvProblems.FooterRow.FindControl("txtSectionHeading"));
    dSource.InsertParameters["SectionHeading"].DefaultValue = tBox.Text;

    // get the has sample selection and set the parameter value
    ddList = (DropDownList)
    (gvProblems.FooterRow.FindControl("selHasCSSample"));
    dSource.InsertParameters["HasCSExample"].DefaultValue =
          ddList.SelectedItem.Value;

    // insert the row in the database
    dSource.Insert();
```

```
    }
   } // gvProblems_RowCommand
  } //  CH02GridViewWithInsertCS
}
```

Recipe 2.19. Formatting Columnar Data in a GridView

Problem

You need to format dates and numbers in your `GridView` controls.

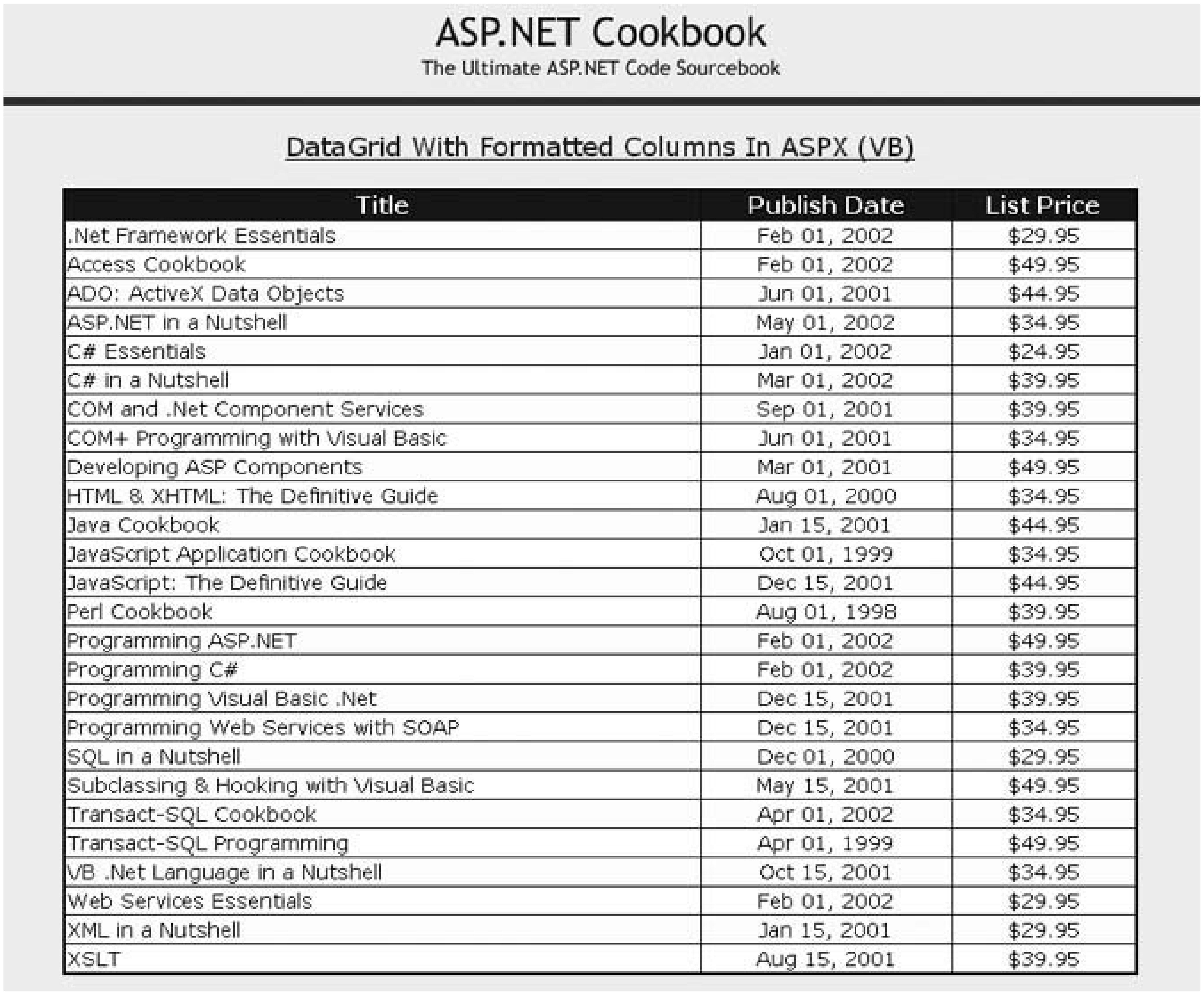
Solution

Use the `DataFormatString` attribute of the `asp:BoundField` element:

1. Within the *.aspx* file that contains the `GridView` control, add a `BoundField` element with the appropriate `DataFormatString` attribute for each column you want to format.
2. If the `DataFormatString` does not provide the flexibility you need to format your data, use the `ItemDataBound` event instead to gain greater flexibility.

Figure 2-20 shows the appearance of an example `GridView` with the Publish Date and List Price columns formatted for dates and currency, respectively. Examples 2-48 , 2-49 through 2-50 show the *.aspx* and code-behind files for an application that produces this result.

Figure 2-20. Formatting columnar data in a GridView output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

DataGrid With Formatted Columns In ASPX (VB)

Title	Publish Date	List Price
.Net Framework Essentials	Feb 01, 2002	$29.95
Access Cookbook	Feb 01, 2002	$49.95
ADO: ActiveX Data Objects	Jun 01, 2001	$44.95
ASP.NET in a Nutshell	May 01, 2002	$34.95
C# Essentials	Jan 01, 2002	$24.95
C# in a Nutshell	Mar 01, 2002	$39.95
COM and .Net Component Services	Sep 01, 2001	$39.95
COM+ Programming with Visual Basic	Jun 01, 2001	$34.95
Developing ASP Components	Mar 01, 2001	$49.95
HTML & XHTML: The Definitive Guide	Aug 01, 2000	$34.95
Java Cookbook	Jan 15, 2001	$44.95
JavaScript Application Cookbook	Oct 01, 1999	$34.95
JavaScript: The Definitive Guide	Dec 15, 2001	$44.95
Perl Cookbook	Aug 01, 1998	$39.95
Programming ASP.NET	Feb 01, 2002	$49.95
Programming C#	Feb 01, 2002	$39.95
Programming Visual Basic .Net	Dec 15, 2001	$39.95
Programming Web Services with SOAP	Dec 15, 2001	$34.95
SQL in a Nutshell	Dec 01, 2000	$29.95
Subclassing & Hooking with Visual Basic	May 15, 2001	$49.95
Transact-SQL Cookbook	Apr 01, 2002	$34.95
Transact-SQL Programming	Apr 01, 1999	$49.95
VB .Net Language in a Nutshell	Oct 15, 2001	$34.95
Web Services Essentials	Feb 01, 2002	$29.95
XML in a Nutshell	Jan 15, 2001	$29.95
XSLT	Aug 15, 2001	$39.95

Discussion

The formatting of dates and numbers in a `GridView` is performed with the `DataFormatString` attribute of the `asp:BoundField` element. The general format of the formatting string is {*`A:B`* }, where *`A`* is the zero-based index number of the property the format applies to (this is generally`0`) and *`B`* specifies the format.

Numeric formats can be any of the following. Most numeric formats can be followed by an integer defining the number of decimal places displayed.

Table 2-2.

Format character	Description
`C`	Displays numeric values in currency format

Format character	Description
`D`	Displays numeric values in decimal format
`E`	Displays numeric values in scientific (exponential) format
`F`	Displays numeric values in fixed format
`G`	Displays numeric values in general format
`N`	Displays numeric values in number format
`X`	Displays numeric values in hexadecimal format

Time/date formats can be any combination of the following:

Table 2-3.

Format character	Associated property/description	Example format pattern (en-US)
`d`	*ShortDatePattern*	`MM/dd/yyyy`
`D`	*LongDatePattern*	`dddd, dd MMMM yyyy`
`f`	Full date and time (long date and short time)	`dddd, dd MMMM yyyy HH:mm`
`F`	*FullDateTimePattern* (long date and long time)	`dddd, dd MMMM yyyy HH:mm:ss`
`g`	General (short date and short time)	`MM/dd/yyyy HH:mm`
`G`	General (short date and long time)	`MM/dd/yyyy HH:mm:ss`
`M, M`	*MonthDayPattern*	`MMMM dd`
`r, R`	*RFC1123Pattern*	`ddd, dd MMM yyyy HH':'mm':'ss 'GMT'`
`S`	*SortableDateTimePattern* (based on ISO 8601) using local time	`yyyy'-'MM'-'dd'T'HH':'mm':'ss`
`t`	*ShortTimePattern*	`HH:mm`
`T`	*LongTimePattern*	`HH:mm:ss`
`u`	*UniversalSortableDateTimePattern* using universal time	`yyyy'-'MM'-'dd HH':'mm':'ss'Z'`
`U`	Full date and time (long date and long time) using universal time	`dddd, dd MMMM yyyy HH:mm:ss`
`y, Y`	*YearMonthPattern*	`yyyy MMMM`

Formatting can be applied when using data binding to any other controlincluding text boxes, repeaters, and the likeby passing the same format string described in this recipe as the second parameter of the`Eval` method. For example:

```
Eval(" PublishDate ",
  "{0:MMM dd, yyyy}")
Eval(" ListPrice",
  "{0:C2}")
```

Formatting data in this manner can be costly in terms of performance. A less costly approach is shown next.

If the `DataFormatString` does not provide the flexibility you need to format your data, the `RowDataBound` event can be used to provide total flexibility in the data presented. As with most events in ASP.NET, the `RowDataBound` event is passed two parameters. The first argument is the sender of the event. In this case, it will be the`GridView` . The second argument is the event arguments. This parameter (by default named e) provides a reference to the item that has been data bound. By using this argument, each column in the row that has been data bound can be accessed and the data formatted as required. There are almost no limits to the reformatting that can be done using the `RowDataBound` event. The following code provides an example of using the`RowDataBound` event to format the Publish Date and List Price columns in our example:

```
Protected Sub gvBooks_RowDataBound(ByVal sender As Object, _
   ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs)
 Const DATE_PUBLISHED_COL As Integer = 1
 Const LIST_PRICE_COL As Integer = 2


 Dim cell As TableCell
 Dim datePublished As Date
 Dim listPrice As Single

 'make sure the item data bound is a data row since this event is also
 'called for the header, footer, pager, etc. and no formatting is
 'required for these items
 If (e.Row.RowType = DataControlRowType.DataRow) Then
  'get the date published that was placed in the GridView during data
  'binding and reformat it as required
  cell = CType(e.Row.Controls(DATE_PUBLISHED_COL), _
   DataControlFieldCell)
  datePublished = CType(cell.Text, _
    Date)
```

```
      cell.Text = datePublished.ToString("MMM dd, yyyy")

      'get the list price that was placed in the GridView during data
      'binding and reformat it as required
      cell = CType(e.Row.Controls(LIST_PRICE_COL), _
       DataControlFieldCell)
      listPrice = CType(cell.Text, _
        Single)
      cell.Text = listPrice.ToString("C2")
    End If
  End Sub 'gvBooks_RowDataBound
```

```
  protected void gvBooks_RowDataBound(Object sender,
     System.Web.UI.WebControls.GridViewRowEventArgs e)
  {
    const int DATE_PUBLISHED_COL = 1;
    const int LIST_PRICE_COL = 2;

    TableCell cell;
    DateTime datePublished;
    Single listPrice;

    // make sure the item data bound is adata row since this event is also
    // called for the header, footer, pager, etc. and no formatting is
    // required for these items
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
      // get the date published that was placed in the GridView during data
      // binding and reformat it as required
      cell = (DataControlFieldCell)(e.Row.Controls[DATE_PUBLISHED_COL]);
      datePublished = Convert.ToDateTime(cell.Text);
      cell.Text = datePublished.ToString("MMM dd, yyyy");

      // get the list price that was placed in the GridView during data
      // binding and reformat it as required
      cell = (DataControlFieldCell)(e.Row.Controls[LIST_PRICE_COL]);
      listPrice = Convert.ToSingle(cell.Text);
      cell.Text = listPrice.ToString("C2");
    }
  } //gvBooks_RowDataBound
```

Remove any `DataFormatString` properties from the `BoundField` elements when using this method of formatting. The additional data conversions and formatting will result in a performance hit as well as potential confusion if the formatting is coded differently.

See Also

Search *Standard Numeric Format Strings* in the MSDN Library.

Example 2-48. Formatting columnar data in a GridView (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02GridViewWithFormattedColumnsVB1.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02GridViewWithFormattedColumnsVB1"
 Title="GridView With Formatted Columns - ASPX" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  GridView With Formatted Columns In ASPX (VB)
 </div>
 <asp:GridView ID="gvBooks" Runat="Server"
    AllowPaging="false"
    AllowSorting="false"
    AutoGenerateColumns="false"
     BorderColor="#000080"
    BorderStyle="Solid"
     BorderWidth="2px"
    Caption=""
    HorizontalAlign="Center"
    Width="90%" >
  <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
  <RowStyle cssClass="tableCellNormal" />
  <AlternatingRowStyle cssClass="tableCellAlternating" />
  <Columns>
   <asp:BoundField DataField="Title"
    HeaderText="Title" />
   <asp:BoundField HeaderText="Publish Date"
    DataField="PublishDate"
    ItemStyle-HorizontalAlign="Center"
    DataFormatString="{0:MMM dd, yyyy}" />
   <asp:BoundField HeaderText="List Price"
    DataField="ListPrice"
    ItemStyle-HorizontalAlign="Center"
    DataFormatString="{0:C2}" />
  </Columns>
 </asp:GridView>
</asp:Content>
```

Example 2-49. Formatting columnar data in a GridView code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02GridViewWithFormattedColumnsVB1.aspx
 ''' </summary>
 Partial  Class  CH02GridViewWithFormattedColumnsVB1
  Inherits System.Web.UI.Page

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
   Dim dSource As SqlDataSource = Nothing

   If (Not Page.IsPostBack) Then
    'configure the data source to get the data from the database
    dSource = New SqlDataSource()
    dSource.ConnectionString = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dSource.DataSourceMode = SqlDataSourceMode.DataSet
    dSource.ProviderName = "System.Data.OleDb"
    dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " & _
       "FROM Book " & _
       "ORDER BY Title"

    'set the source of the data for the gridview control and bind it
    gvBooks.DataSource = dSource
    gvBooks.DataBind()
   End If
  End Sub 'Page_Load
 End  Class  'CH02GridViewWithFormattedColumnsVB1
End Namespace
```

Example 2-50. Formatting columnar data in a GridView code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples

{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02GridViewWithFormattedColumnsCS1.aspx
 /// </summary>
 public partial class CH02GridViewWithFormattedColumnsCS1
  : System.Web.UI.Page
 {
  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   SqlDataSource dSource = null;

   if (!Page.IsPostBack)
   {
    // configure the data source to get the data from the database
    dSource = new SqlDataSource();
    dSource.ConnectionString = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dSource.DataSourceMode = SqlDataSourceMode.DataReader;
    dSource.ProviderName = "System.Data.OleDb";
    dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " +
       "FROM Book " +
       "ORDER BY Title";

    // set the source of the data for the gridview control and bind it
    gvBooks.DataSource = dSource;
    gvBooks.DataBind();
   }
  } // Page_Load
 } //  CH02GridViewWithFormattedColumnsCS1
}
```

PREV

PREV

Recipe 2.20. Allowing Selection Anywhere Within a GridView

Problem

You are implementing a `GridView` that requires selection of a row, but you do not want to have a Select button in every row of your `GridView` . What you really want is to allow the user to click anywhere within a row, like in a classic Windows application.

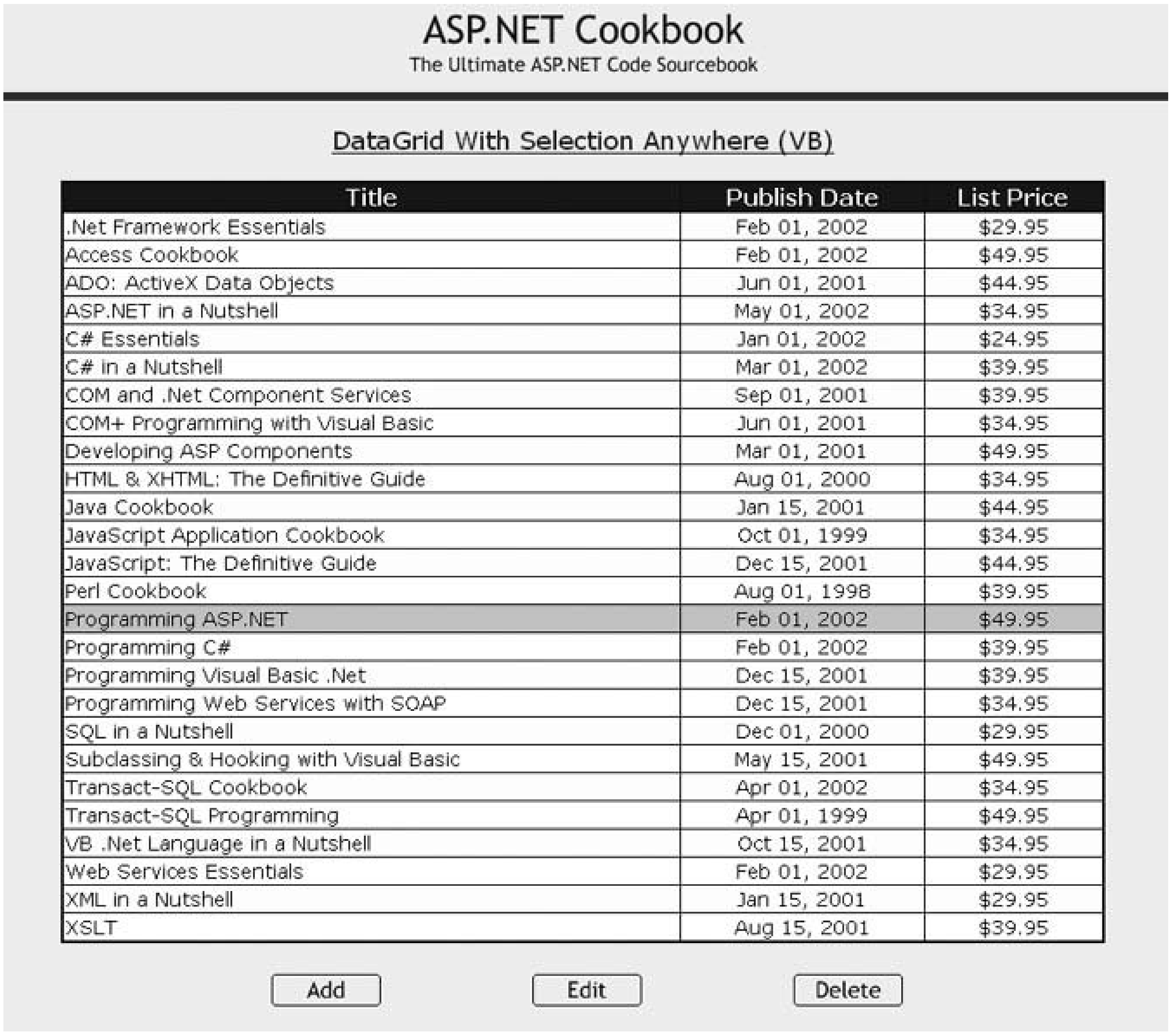
Solution

To every row in the `GridView` , add a hidden Select button along with an `onclick` event that performs the same action as if the hidden Select button were clicked:

1. Add a hidden `ButtonField` to the `GridView` .
2. Set the `ButtonType` attribute to `Link` so a hidden hyperlinked Select button is rendered in every row.
3. In the `RowDataBound` event, add an `onclick` event to the `GridView` row that performs the same action as clicking the hidden Select button.

The approach produces output like that shown in Figure 2-21. Examples 2-51 , 2-52 through 2-53 show the *.aspx* and code-behind files for the application that produces this result.

Figure 2-21. Output of GridView allowing selection anywhere


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
DataGrid With Selection Anywhere (VB)
Add
Edit
Delete

Title	Publish Date	List Price
.Net Framework Essentials	Feb 01, 2002	$29.95
Access Cookbook	Feb 01, 2002	$49.95
ADO: ActiveX Data Objects	Jun 01, 2001	$44.95
ASP.NET in a Nutshell	May 01, 2002	$34.95
C# Essentials	Jan 01, 2002	$24.95
C# in a Nutshell	Mar 01, 2002	$39.95
COM and .Net Component Services	Sep 01, 2001	$39.95
COM+ Programming with Visual Basic	Jun 01, 2001	$34.95
Developing ASP Components	Mar 01, 2001	$49.95
HTML & XHTML: The Definitive Guide	Aug 01, 2000	$34.95
Java Cookbook	Jan 15, 2001	$44.95
JavaScript Application Cookbook	Oct 01, 1999	$34.95
JavaScript: The Definitive Guide	Dec 15, 2001	$44.95
Perl Cookbook	Aug 01, 1998	$39.95
Programming ASP.NET	Feb 01, 2002	$49.95
Programming C#	Feb 01, 2002	$39.95
Programming Visual Basic .Net	Dec 15, 2001	$39.95
Programming Web Services with SOAP	Dec 15, 2001	$34.95
SQL in a Nutshell	Dec 01, 2000	$29.95
Subclassing & Hooking with Visual Basic	May 15, 2001	$49.95
Transact-SQL Cookbook	Apr 01, 2002	$34.95
Transact-SQL Programming	Apr 01, 1999	$49.95
VB .Net Language in a Nutshell	Oct 15, 2001	$34.95
Web Services Essentials	Feb 01, 2002	$29.95
XML in a Nutshell	Jan 15, 2001	$29.95
XSLT	Aug 15, 2001	$39.95

Discussion

To allow selection of a row of data by clicking on it, you create a`GridView` in the usual fashion but add a hidden `ButtonField` . The `ButtonType` attribute is set to `Link` , and the `CommandName` attribute is set to `Select` . This causes the `GridView` to be rendered with a hidden hyperlinked Select button in every row.

```
<Columns>
 <asp:ButtonField ButtonType="Link"
   Visible="False"
   CommandName="Select" />
 …
</Columns>
```

In the code-behind, the GridView control's RowDataBound event handler (gvBooks_RowDataBound) is used to expand the functionality of the hidden Select button to encompass the entire row. This method is called for every row of the GridView , including the header and footer, so the item type must be checked to see if this event applies to a given data row.

When the event applies to a data row, you must first get a reference to the hidden Select button in the row. The LINK_BUTTON_COLUMN and LINK_BUTTON_CONTROL constants are used to avoid so-called "magic numbers" (hardcoded numbers that seem to appear out of nowhere in the code) and to make the code more maintainable.

Next, some client-side JavaScript is added to a row's hidden hyperlinked Select button. Its two purposes are to handle the onclick event for the row in the GridView that has been data bound and to perform a call to__doPostBack . The JavaScript is added to the GridView row's Attributes collection using the Add method, whose parameters are the name of the event we want to add to the control and the name of the function (along with its parameters) to be executed when the event occurs.

The Page 's GetPostBackClientHyperlink method is used to get the name of the clientside function created for the hidden Select button in the row being processed. It returns the name of the event method along with the required parameters. For the first row in ourGridView , for example, the GetPostBackClientHyperlink method returns javascript:__doPostBack('ctl00$PageBody$gvBooks$ctl02$ctl00 ','').

Effectively, this adds an onclick event to all the table rows, which causes the method __doPostBack to be called anytime the user clicks on a data row in the grid. Because thisonclick event is identical to the event created for the hidden Select button in the row, thepostback is processed as a select event, thereby setting the SelectedIndex of the GridView to the clicked row.

Be aware that the selection of a row using this method requires a round trip to the server to perform the selection.

By default, ASP.NET 2.0 validates that events are raised only by controls that are enabled and are in parent hierarchies that are also enabled. Since the technique described in this recipe uses an event posted back from a hidden button, an exception will be thrown unless event validation is disabled by setting the EnableEventValidation attribute of the @ Page directive to false :

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.
master"
 AutoEventWireup="false"
  CodeFile="CH02GridViewWithSelectionAnywhereVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.
CH02GridViewWithSelectionAnywhereVB"
 EnableEventValidation="false"
 Title="GridView With Selection Anywhere" %>
```

This example shows the use of Add, Edit, and Delete buttons below the`GridView` , which is typical of a scenario where a row is selected and then a desired action is performed on it. The methods for the Add, Edit, and Delete events were included in this recipe but were left empty to keep the code down to a reasonable size.

Example 2-51. GridView allowing selection anywhere (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH02GridViewWithSelectionAnywhereVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH02GridViewWithSelectionAnywhereVB"
  EnableEventValidation="false"
  Title="GridView With Selection Anywhere" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
<div align="center" class="pageHeading">
 GridView With Selection Anywhere (VB)
</div>
<asp:GridView ID="gvBooks" Runat="Server"
     AllowPaging="false"
     AllowSorting="false"
     AutoGenerateColumns="false"
     BorderColor="#000080"
     BorderStyle="Solid"
     BorderWidth="2px"
     Caption=""
     HorizontalAlign="Center"
     Width="90%"
     OnRowDataBound="gvBooks_RowDataBound" >
<HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
<RowStyle cssClass="tableCellNormal" />
<AlternatingRowStyle cssClass="tableCellAlternating" />
<SelectedRowStyle CssClass="tableCellSelected" />
<Columns>
  <asp:ButtonField ButtonType="Link"
      Visible="False"
      CommandName="Select" />
  <asp:BoundField HeaderText="Title"
      DataField="Title"
      ItemStyle-HorizontalAlign="Left" />
  <asp:BoundField HeaderText="Publish Date"
      DataField="PublishDate"
      ItemStyle-HorizontalAlign="Center"
      DataFormatString="{0:MMM dd, yyyy}"/>
  <asp:BoundField HeaderText="List Price"
      DataField="ListPrice"
      ItemStyle-HorizontalAlign="Center"
      DataFormatString="{0:C2}"/>
</Columns>
```

```
   </asp:GridView>
   <br />
    <table width="40%" border="0" align="center">
 <tr>
  <td align="center">

  <input id="btnAdd" runat="server" type="button"
    value="Add"
    onserverclick="btnAdd_ServerClick">
 </td>
 <td align="center">
  <input id="btnEdit" runat="server" type="button"
    value="Edit"
    onserverclick="btnEdit_ServerClick">
 </td>
 <td align="center">
  <input id="btnDelete" runat="server" type="button"
    value="Delete"
    onserverclick="btnDelete_ServerClick">
  </td>
   </tr>
  </table>
</asp:Content>
```

Example 2-52. GridView allowing selection anywhere code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02GridViewWithSelectionAnywhereVB.aspx
 ''' </summary>
 Partial  Class  CH02GridViewWithSelectionAnywhereVB
  Inherits System.Web.UI.Page

   '''**************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
     If (Not Page.IsPostBack) Then
        bindData( )
     End If
   End Sub 'Page_Load

 '''*************************************************************************
 ''' <summary>
 ''' This routine is the event handler that is called when the Add button
 ''' is clicked.
 ''' </summary>
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnAdd_ServerClick(ByVal sender As Object, _
       ByVal e As System.EventArgs) _
    'place code here to perform Add operations
  End Sub 'btnAdd_ServerClick

  '''*************************************************************************
  ''' <summary>
  ''' This routine is the event handler that is called when the Edit button
  ''' is clicked.
  ''' </summary>
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnEdit_ServerClick(ByVal sender As Object, _
         ByVal e As System.EventArgs) _
  'place code here to perform Edit operations
  End Sub 'btnEdit_ServerClick

  '''*************************************************************************
  ''' <summary>
  ''' This routine is the event handler that is called when the Delete button
  ''' is clicked.
  ''' </summary>
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnDelete_ServerClick(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
  'place code here to perform Delete operations
  End Sub 'btnDelete_ServerClick

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the GridView's row data
  ''' bound event. It is responsible for formatting the data in the
  ''' columns of the GridView
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub gvBooks_RowDataBound(ByVal sender As Object, ByVal e As _
    System.Web.UI.WebControls.GridViewRowEventArgs)
  'GridView column containing link button defined on ASPX page
  Const LINK_BUTTON_COLUMN As Integer = 0
  'index of link button control in the link button column
  Const LINK_BUTTON_CONTROL As Integer = 0

  Dim button As LinkButton

  'check the type of item that was databound and only take action if it
  'was a row in the GridView
  If (e.Row.RowType = DataControlRowType.DataRow) Then
    'the item that was bound is a so get a reference to the link button
    'column defined in the Columns property of the GridView (in the aspx
    'file) and add an event handler for the the onclick event for this
    'entire row. This will make clicking anywhere in the row select
    'the row.
    'NOTE: This is tightly coupled to the definition of the bound columns
    '  in the aspx page.
    button = _
      CType(e.Row.Cells(LINK_BUTTON_COLUMN).Controls(LINK_BUTTON_CONTROL), _
    LinkButton)
   e.Row.Attributes.Add("onclick", _
       ClientScript.GetPostBackClientHyperlink(button, ""))
    End If
 End Sub 'gvBooks_RowDataBound

 '''***************************************************************************
 ''' <summary>
 ''' This routine queries the database for the data to displayed and binds
 ''' it to the GridView
 ''' </summary>
 Private Sub bindData( )
   Dim dSource As SqlDataSource = Nothing

   'configure the data source to get the data from the database
   dSource = New SqlDataSource( )
   dSource.ConnectionString = ConfigurationManager. _
      ConnectionStrings("dbConnectionString").ConnectionString
   dSource.DataSourceMode = SqlDataSourceMode.DataSet
   dSource.ProviderName = "System.Data.OleDb"
   dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " & _
      "FROM Book " & _
      "ORDER BY Title"

   'set the source of the data for the gridview control and bind it
    gvBooks.DataSource = dSource
    gvBooks.DataBind( )

  'select first item in the gridview
```

```
   gvBooks.SelectedIndex = 0
    End Sub 'bindData
 End  Class  'CH02GridViewWithSelectionAnywhereVB
End Namespace
```

Example 2-53. GridView allowing selection anywhere code-behind (.cs)

```

using System;
using System.Configuration;

using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02GridViewWithSelectionAnywhereCS.aspx
 /// </summary>
 public  partial  class  CH02GridViewWithSelectionAnywhereCS
   : System.Web.UI.Page
 {
   ///***************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   if (!Page.IsPostBack)
   {
    bindData( );
   }
  } // Page_Load

   ///***************************************************************************
  /// <summary>
  /// This routine is the event handler that is called when the Add button
  /// is clicked.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnAdd_ServerClick(Object sender,
         System.EventArgs e)
  {
   // place code here to perform Add operations
  } // btnAdd_ServerClick

  ///***********************************************************************
  /// <summary>
  /// This routine is the event handler that is called when the Edit button
  /// is clicked.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnEdit_ServerClick(Object sender,
          System.EventArgs e)

  {
   // place code here to perform Edit operations
  } // btnEdit_ServerClick

  ///***********************************************************************
  /// <summary>
  /// This routine is the event handler that is called when the Delete button
  /// is clicked.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnDelete_ServerClick(Object sender,
         System.EventArgs e)
  {
   // place code here to perform Delete operations
  } // btnDelete_ServerClick

  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the GridView's row created
  /// event. It is responsible for setting the icon in the header row to
  /// indicate the current sort column and sort order
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void gvBooks_RowDataBound(Object sender,
    System.Web.UI.WebControls.GridViewRowEventArgs e)
  {
   // gridview column containing link button defined on ASPX page
   const int LINK_BUTTON_COLUMN = 0;
   // index of link button control in the link button column
   const int LINK_BUTTON_CONTROL = 0;

   LinkButton button = null;
```

```
   // check the type of row that was databound and only take action if it
   // was a data row
   if (e.Row.RowType == DataControlRowType.DataRow)
   {
    // the item that was bound is a data row so get a reference to the
    // link button column defined in the Columns property of the datagrid
    // (in the aspx page) and add an event handler for the the onclick
    // event for this entire row. This will make clicking anywhere in the
    // row select the row.
    // NOTE: This is tightly coupled to the definition of the bound
    //  columns in the aspx page.
    button =
  (LinkButton)(e.Row.Cells[LINK_BUTTON_COLUMN].Controls[LINK_BUTTON_CONTROL]);
    e.Row.Attributes.Add("onclick",
      Page.GetPostBackClientHyperlink(button, ""));

   }
  } //gvBooks_RowDataBound

  ///*************************************************************************
  /// <summary>
  /// This routine queries the database for the data to displayed and
  /// binds it to the GridView
  /// </summary>
  private void bindData( )
  {
   SqlDataSource dSource = null;

   // configure the data source to get the data from the database
   dSource = new SqlDataSource( );
   dSource.ConnectionString = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
   dSource.DataSourceMode = SqlDataSourceMode.DataReader;
   dSource.ProviderName = "System.Data.OleDb";
   dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " +
          "FROM Book " +
       "ORDER BY Title";

   // set the source of the data for the gridview control and bind it
   gvBooks.DataSource = dSource;
   gvBooks.DataBind( );

   // select first item in the gridview
    gvBooks.SelectedIndex = 0;
  } // bindData
 } //  CH02GridViewWithSelectionAnywhereCS
}
```

PREV

Recipe 2.21. Adding a Delete Confirmation Pop-Up

Problem

You want to add to a `GridView` row a confirmation pop-up that appears whenever a user tries to delete a row in the `GridView` .

Solution

Add a Select button to each row of the `GridView` and a Delete button below the `GridView` . Whenever the Delete button is clicked, execute some client-side script that displays the confirmation pop-up, followed by some server-side code that performs the actual deletion.

In the *.aspx* file:

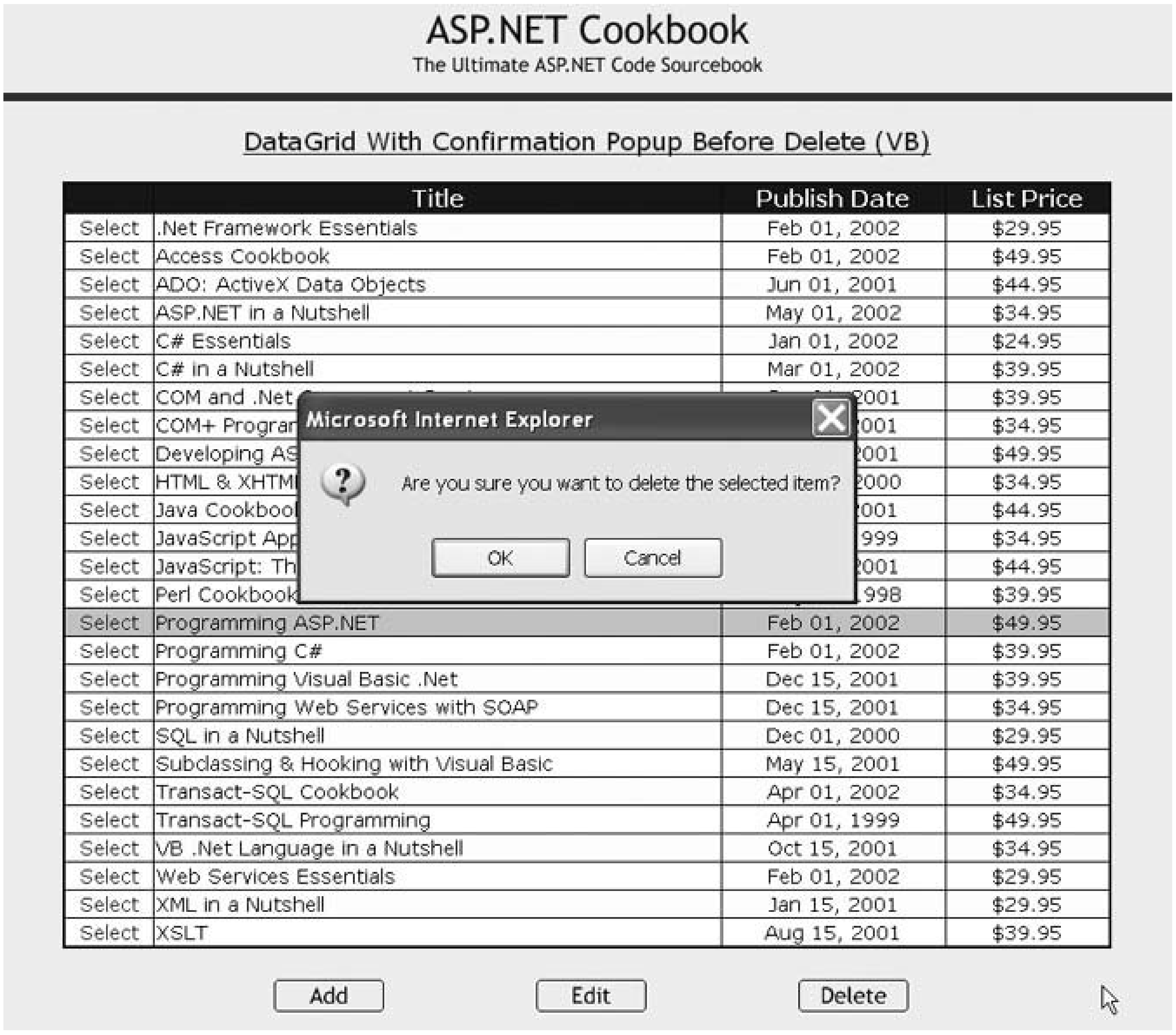
1. Create an extra button column in the `GridView` to display a Select button.
2. Add a Delete button below the `GridView` .

In the code-behind class for the page, use the .NET language of your choice to:

1. Register the client-side script to be executed when the Delete button is clicked.
2. Add an attribute to the Delete button that calls the delete script when the Delete button is clicked.

Figure 2-22 shows a `GridView` with this solution implemented. Examples 2-54 , 2-55 through 2-56 show the *.aspx* and code-behind files for the application that produces this result.

Figure 2-22. Confirmation pop-up before deletion in a GridView output



ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

DataGrid With Confirmation Popup Before Delete (VB)

	Title	Publish Date	List Price
Select	.Net Framework Essentials	Feb 01, 2002	$29.95
Select	Access Cookbook	Feb 01, 2002	$49.95
Select	ADO: ActiveX Data Objects	Jun 01, 2001	$44.95
Select	ASP.NET in a Nutshell	May 01, 2002	$34.95
Select	C# Essentials	Jan 01, 2002	$24.95
Select	C# in a Nutshell	Mar 01, 2002	$39.95
Select	COM and .Net	2001	$39.95
Select	COM+ Progran	001	$34.95
Select	Developing AS	2001	$49.95
Select	HTML & XHTMI	2000	$34.95
Select	Java Cookbool	001	$44.95
Select	JavaScript App	999	$34.95
Select	JavaScript: Th	2001	$44.95
Select	Perl Cookbook	998	$39.95
Select	Programming ASP.NET	Feb 01, 2002	$49.95
Select	Programming C#	Feb 01, 2002	$39.95
Select	Programming Visual Basic .Net	Dec 15, 2001	$39.95
Select	Programming Web Services with SOAP	Dec 15, 2001	$34.95
Select	SQL in a Nutshell	Dec 01, 2000	$29.95
Select	Subclassing & Hooking with Visual Basic	May 15, 2001	$49.95
Select	Transact-SQL Cookbook	Apr 01, 2002	$34.95
Select	Transact-SQL Programming	Apr 01, 1999	$49.95
Select	VB .Net Language in a Nutshell	Oct 15, 2001	$34.95
Select	Web Services Essentials	Feb 01, 2002	$29.95
Select	XML in a Nutshell	Jan 15, 2001	$29.95
Select	XSLT	Aug 15, 2001	$39.95

Microsoft Internet Explorer

Are you sure you want to delete the selected item?

Discussion

To display a confirmation pop-up when a user attempts to delete a row in a data table, you create a `GridView` in the same way you have done throughout this chapter, except that you add a button column to allow for row selection. Setting the `ButtonType` to "`Link` " outputs a hyperlink for selecting the row. (The `ButtonType` can instead be set to "`Button` " to output an `HTMLInputButton` control or "`Image` " to output an `HTMLImageButton` control.) The `CommandName` defines the action to be taken when the button is clicked, and the `Text` attribute defines the text that will be output for the button. (The "select anywhere approach" described in Recipe 2.19 can be used here instead.)

```
<Columns>
 <asp:ButtonField ButtonType="Link"
   CommandName="Select"
   Text="Select" />
  …
```

```
</Columns>
```

From here on, it's easiest to explain the remaining steps of this recipe in the context of our actual example application. In the `Page_Load` method of the code-behind, the client-side script block to be executed when the Delete button is clicked is created and registered with the page. The `ClientScript.IsClientScriptBlockRegistered` method is used to ensure that the script block is not registered more than once on the page. The `ClientScript.RegisterClientScriptBlock` method is used to output the script block in the page when the page is rendered. This method causes the script to output immediately after the opening `Form` tag. If you prefer the script to be output immediately *before* the `Form` *end* tag, the `ClientScript.RegisterStartupScript` method can be used instead.

In ASP.NET 1.x, the `RegisterClientScriptBlock` was part of the Page class and had only two parameters: the name of the script (the key) and the script that was to be output. In ASP.NET 2.0, the `RegisterClientScriptBlock` method has been moved to the new `ClientScript` class and two new parameters have been added. The first parameter is a type and can be used in conjunction with the key to identify the script uniquely. This helps avoid the problem of scripts with the same name being output from other controls, resulting in an improperly executing page. The type is generally set to th type of the controlling container such as the page class or the user control class.

The second and third parameters of the `RegisterClientScriptBlock` are the key and script, as within ASP.NET 1.x. The fourth parameter is a `Boolean` that when set `true` will cause the script tags to be output automatically so your script block does not have to contain the tags as it did in ASP.NET 1.x.

The client-side script that is output to the browser is shown here:

```
<script type="text/javascript">
<!-
function beforeDelete()
{return(confirm('Are you sure you want to delete the selected item?'));}
// -->
</script>
```

The code that outputs the client-side script block must be executed every time the page is rendered, including `postbacks` , because the registered script blocks are not persisted in the `Page` object.

After creating and registering the client script block, an attribute is added to the Delete button contro to cause the client script block to be executed when the button is clicked. The resulting HTML for the delete button is shown here:

```
<input type="submit"
   name="ctl00$PageBody$btnDelete"
  value="Delete"
  onclick="return(beforeDelete());"
```

```
    id="ctl00_PageBody_btnDelete" />
```

When the user clicks the Delete button, the `beforeDelete` function is called in the client-side code. The `beforeDelete` function outputs a standard HTML confirmation dialog box with the message, "Are you sure you want to delete the selected item?" If the user clicks the Cancel button, the function returns `False` , effectively canceling the `postback` of the page. If the user clicks the OK button, the function returns `TRue` , allowing the page to be posted back to the server.

A server-side event handler (`btnDelete_ServerClick`) is added to the code-behind to handle the Delete button's server-side click event. In this method, a check is performed to ensure a row is selected, and then the required deletion code for your application is processed.

The JavaScript registered in the code-behind and the attribute added to the Delete button can also be placed directly in the *.aspx* file. This was not done in this example, though, in the spirit of keeping all code in the code-behind and all presentation aspects in the *.aspx* file, a highly recommended practice. By using this approach, you can create a library of client-side scripts once and reuse them many times throughout your applications.

Using a radio button for row selection instead of a Select button would be preferable, but a bug in Releases 1.0, 1.1, and 2.0 of ASP.NET makes it difficult. The problem is caused by a unique name and group name being generated for every control in the grid, thus placing the radio buttons on each row in a different group. This has the unfortunate consequence of allowing a user to select multiple radio buttons at the same time. For details of the bug, see Knowledge Base article Q316495 on Microsoft's MSDN web site (http://msdn.microsoft.com).

See Also

Recipe 2.19

Example 2-54. Confirmation pop-up before deletion in a GridView (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02GridViewWithConfirmBeforeDeleteVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02GridViewWithConfirmBeforeDeleteVB"
 Title="GridView With Confirmation Popup Before Delete" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  GridView With Confirmation Popup Before Delete (VB)
 </div>
 <asp:GridView ID="gvBooks" Runat="Server"
    AllowPaging="false"
    AllowSorting="false"
    AutoGenerateColumns="false"
    BorderColor="#000080"

    BorderStyle="Solid"
    BorderWidth="2px"
    Caption=""
    HorizontalAlign="Center"
    Width="90%" >
 <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
 <RowStyle cssClass="tableCellNormal" />
 <AlternatingRowStyle cssClass="tableCellAlternating" />
 <SelectedRowStyle CssClass="tableCellSelected" />
 <Columns>
   <asp:ButtonField ButtonType="Link"
       CommandName="Select"
       Text="Select" />
   <asp:BoundField HeaderText="Title"
       DataField="Title"
       ItemStyle-HorizontalAlign="Left" />
      <asp:BoundField HeaderText="Publish Date"
       DataField="PublishDate"
       ItemStyle-HorizontalAlign="Center"
       DataFormatString="{0:MMM dd, yyyy}"/>
   <asp:BoundField HeaderText="List Price"
       DataField="ListPrice"
       ItemStyle-HorizontalAlign="Center"
       DataFormatString="{0:C2}"/>
 </Columns>
</asp:GridView>
<br />
<table  width="40%"  border="0"  align="center">
 <tr>
  <td align="center">
   <asp:Button ID="btnAdd" runat="server"
      Text="Add"
      OnClick="btnAdd_ServerClick" />
  </td>
  <td align="center">
   <asp:Button ID="btnEdit" runat="server"
      Text="Edit"
      OnClick="btnEdit_ServerClick" />
  </td>
  <td align="center">
   <asp:Button ID="btnDelete" runat="server"
      Text="Delete"
      OnClick="btnDelete_ServerClick" />
  </td>
   </tr>
 </table>
```

```
</asp:Content>
```

Example 2-55. Confirmation pop-up before deletion in a GridView code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02GridViewWithConfirmBeforeDeleteVB.aspx
 ''' </summary>
 Partial  Class  CH02GridViewWithConfirmBeforeDeleteVB
  Inherits System.Web.UI.Page

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
   Dim scriptBlock As String

   If (Not Page.IsPostBack) Then
    bindData( )
   End If

   'NOTE: The following code must be processed for every rendering of the
   ' page or the client script will not be output when server click
   ' events are processed.

   'create the script block that will execute when the delete
   'button is clicked and register it
   scriptBlock = "function beforeDelete( )" & vbCrLf & _
           "{return(confirm('Are you sure you want to delete " & _
      "the selected item?'));}"
   If (Not ClientScript.IsClientScriptBlockRegistered("deletePromptScript")) Then
```

```
    ClientScript.RegisterClientScriptBlock(Me.GetType( ), _
         "deletePromptScript", _
         scriptBlock, _
         True)
  End If

  'use the OnClientClick property of the button to cause the above
  'script to be executed when the button is clicked
  btnDelete.OnClientClick = "return(beforeDelete( ));"
 End Sub 'Page_Load

 '''***********************************************************************
 ''' <summary>

''' This routine is the event handler that is called when the Add button
''' is clicked.
''' </summary>
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnAdd_ServerClick(ByVal sender As Object, _
      ByVal e As System.EventArgs) _
  'place code here to perform Add operations
End Sub 'btnAdd_ServerClick

'''***********************************************************************
''' <summary>
''' This routine is the event handler that is called when the Edit button
''' is clicked.
''' </summary>
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnEdit_ServerClick(ByVal sender As Object, _
       ByVal e As System.EventArgs) _
   'place code here to perform Edit operations
End Sub 'btnEdit_ServerClick

'''***********************************************************************
''' <summary>
''' This routine is the event handler that is called when the Delete button
''' is clicked.
''' </summary>
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnDelete_ServerClick(ByVal sender As Object, _
      ByVal e As System.EventArgs) _
   'make sure an item is selected
   If (gvBooks.SelectedIndex >= 0) Then
     'place code here to perform Delete operations
   End If
End Sub 'btnDelete_ServerClick

'''***********************************************************************
```

```
 ''' <summary>
 ''' This routine queries the database for the data to displayed and binds
 ''' it to the GridView
 ''' </summary>
 Private Sub bindData( )
  Dim dSource As SqlDataSource = Nothing

  'configure the data source to get the data from the database
  dSource = New SqlDataSource( )
  dSource.ConnectionString = ConfigurationManager. _
   ConnectionStrings("dbConnectionString").ConnectionString
  dSource.DataSourceMode = SqlDataSourceMode.DataSet
  dSource.ProviderName = "System.Data.OleDb"

  dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " & _
     "FROM Book " & _
     "ORDER BY Title"

  'set the source of the data for the gridview control and bind it
  gvBooks.DataSource = dSource
  gvBooks.DataBind( )

  'select first item in the gridview
  gvBooks.SelectedIndex = 0
  End Sub 'bindData
   End  Class  'CH02GridViewWithConfirmBeforeDeleteVB
End Namespace
```

Example 2-56. Confirmation pop-up before deletion in a GridView code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02GridViewWithConfirmBeforeDeleteCS.aspx
 /// </summary>
 public  partial  class  CH02GridViewWithConfirmBeforeDeleteCS
  : System.Web.UI.Page
 {
  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   String scriptBlock;

   if (!Page.IsPostBack)
   {
    bindData( );
   }

   // NOTE: The following code must be processed for every rendering of the
   // page or the client script will not be output when server click
   // events are processed.

   // create the script block that will execute when the delete
   // button is clicked and register it
   scriptBlock = "function beforeDelete( )\n" +
      "{return(confirm('Are you sure you want to delete " +
      "the selected item?'));}\n";
   if (!ClientScript.IsClientScriptBlockRegistered("deletePromptScript"))
   {
    ClientScript.RegisterClientScriptBlock(this.GetType( ),
            "deletePromptScript",
            scriptBlock,
            true);
   }

   // use the OnClientClick property of the button to cause the above
   // script to be executed when the button is clicked
   btnDelete.OnClientClick = "return(beforeDelete( ));";
  } // Page_Load

  ///***********************************************************************
  /// <summary>
  /// This routine is the event handler that is called when the Add button
  /// is clicked.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnAdd_ServerClick(Object sender,
       System.EventArgs e)
  {
   // place code here to perform Add operations
  } // btnAdd_ServerClick

  ///***********************************************************************
```

```
/// <summary>
/// This routine is the event handler that is called when the Edit button
/// is clicked.
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void btnEdit_ServerClick(Object sender,
      System.EventArgs e)
{
 // place code here to perform Edit operations
} // btnEdit_ServerClick

///***************************************************************************
/// <summary>
/// This routine is the event handler that is called when the Delete button
/// is clicked.
/// </summary>
///

/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void btnDelete_ServerClick(Object sender,
         System.EventArgs e)
{
 // make sure an item is selected
 if (gvBooks.SelectedIndex >= 0)
 {
  // place code here to perform Delete operations
 }
} // btnDelete_ServerClick

///***************************************************************************
/// <summary>
/// This routine queries the database for the data to displayed and
/// binds it to the GridView
/// </summary>
private void bindData( )
{
 SqlDataSource dSource = null;

 // configure the data source to get the data from the database
 dSource = new SqlDataSource( );
 dSource.ConnectionString = ConfigurationManager.
  ConnectionStrings["dbConnectionString"].ConnectionString;
 dSource.DataSourceMode = SqlDataSourceMode.DataReader;
 dSource.ProviderName = "System.Data.OleDb";
 dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " +
    "FROM Book " +
    "ORDER BY Title";

 // set the source of the data for the gridview control and bind it
```

```
   gvBooks.DataSource = dSource;
   gvBooks.DataBind( );

   // select first item in the gridview
    gvBooks.SelectedIndex = 0;
  } // bindData
 }  //  CH02GridViewWithConfirmBeforeDeleteCS
}
```

PREV

Recipe 2.22. Displaying a Pop-Up Details Window

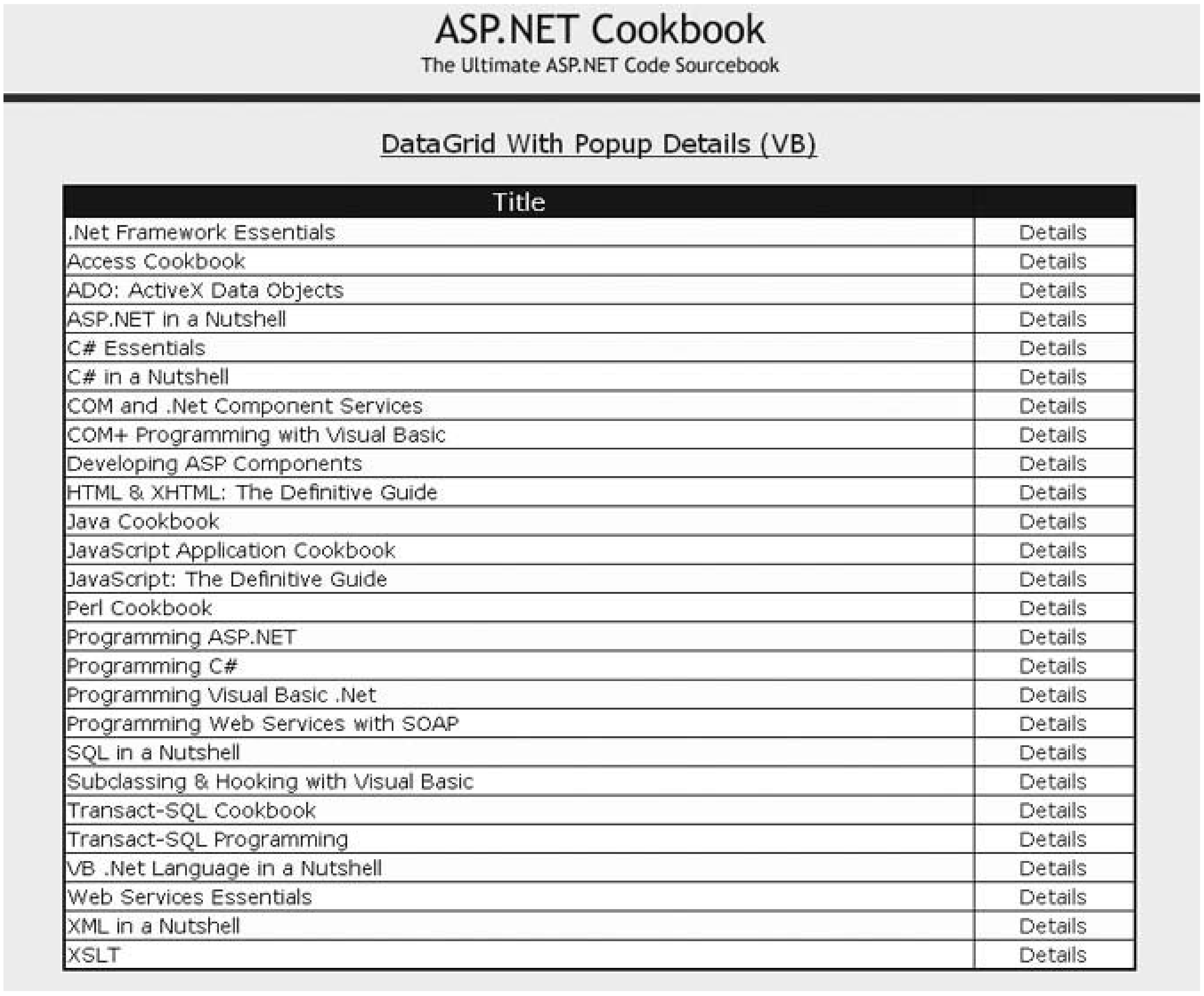
Problem

You want to provide additional details for each row in a `GridView` using a pop-up window.

Solution

Add a Details button to each row in the `GridView` . When the user clicks the button, open a new browser window, obtain the information from the server, and display the detailed information in a pop-up window. An example of the possible output is shown in Figures 2-23 (sample `GridView`) and Figures 2-24 (sample pop-up window output). As with the other recipes in this book, we've implemented a complete application that illustrates this approach. The form and code-behind for the page containing the sample `GridView` is shown in Examples 2-57, 2-58 through 2-59 , and the form and code-behind for the sample pop-up window is shown in Examples 2-60, 2-61 through 2-62 .

Figure 2-23. GridView with pop-up details window output


ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

DataGrid With Popup Details (VB)

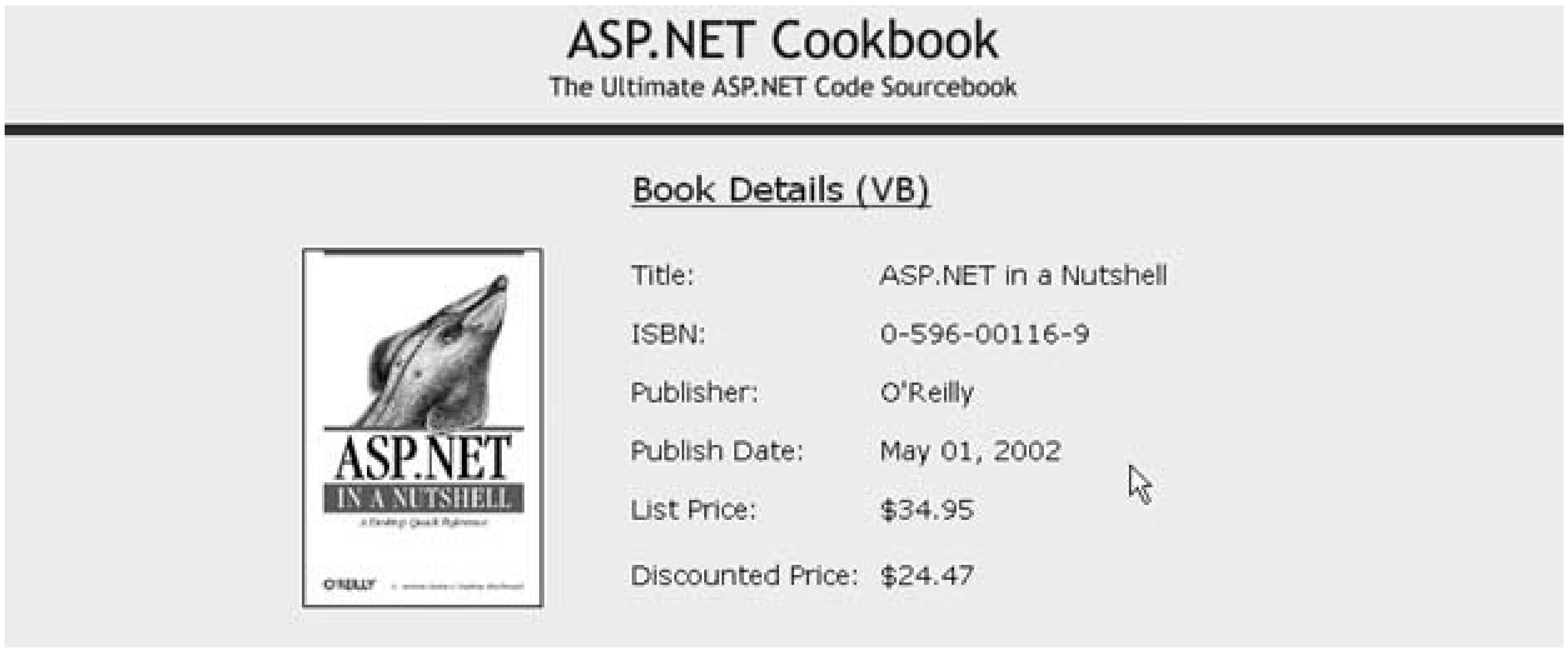
Title	
.Net Framework Essentials	Details
Access Cookbook	Details
ADO: ActiveX Data Objects	Details
ASP.NET in a Nutshell	Details
C# Essentials	Details
C# in a Nutshell	Details
COM and .Net Component Services	Details
COM+ Programming with Visual Basic	Details
Developing ASP Components	Details
HTML & XHTML: The Definitive Guide	Details
Java Cookbook	Details
JavaScript Application Cookbook	Details
JavaScript: The Definitive Guide	Details
Perl Cookbook	Details
Programming ASP.NET	Details
Programming C#	Details
Programming Visual Basic .Net	Details
Programming Web Services with SOAP	Details
SQL in a Nutshell	Details
Subclassing & Hooking with Visual Basic	Details
Transact-SQL Cookbook	Details
Transact-SQL Programming	Details
VB .Net Language in a Nutshell	Details
Web Services Essentials	Details
XML in a Nutshell	Details
XSLT	Details

Discussion

To implement this solution, create a `GridView` in the normal fashion but add a link button column to display a Details link. When the user clicks the Details link within a row of the `GridView` , the browser opens a new window and requests the appropriate page from the server. In the context of our example that implements this solution, a book details page is requested. From here on, the recipe's remaining steps are described in the context of our example because we use techniques that you are likely to find helpful in implementing your own application.

In our example, when the book details page is processed, a book ID is extracted from the query string and is used in the database query to get the detailed data for the specific book as shown in the `Page_Load` method of Examples 2-61 (VB) and 2-62 (C#).

Figure 2-24. Pop-up details window output


ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

Book Details (VB)

Title:	ASP.NET in a Nutshell
ISBN:	0-596-00116-9
Publisher:	O'Reilly
Publish Date:	May 01, 2002
List Price:	$34.95
Discounted Price:	$24.47

When building a Details link in the *.aspx* file, an HTML anchor tag is placed in the `ItemTemplate` for the column. (The purpose of the anchor tag is to request the details page when the associated link button is clicked.) The `target` property of the HTML anchor is set to `_blank` , causing a new browser window to open when the link is clicked.

The `Page_Load` method in the code-behind is nearly identical to that used in other recipes with one change. The lines of code shown next are added to populate the `DataKeyNames` collection of the `GridView` with the primary key values for the rows being displayed. This causes the `GridView` to keep track of the primary key value for each row without our having to output the data in a hidden column. These values are needed later to display the book details.

```
dataKeys(0) = "BookID"
gvBooks.DataKeyNames = dataKeys
```

```
dataKeys = new string[1] {"BookID"};
gvBooks.DataKeyNames = dataKeys;
```

The `GridView` control's `RowDataBound` event is used to set the `href` value for the "details" HTML anchors added to the `GridView` . Because this event is called independently for every row in the `GridView` , the item type must be checked to see if this event applies for a given data row.

When the event does apply to a data row, the first thing we must do is get the ID of the book being displayed in the row as shown here:

```
bookID  =  CInt(gvBooks.DataKeys(e.Row.RowIndex).Item(0))
bookID = (int)(gvBooks.DataKeys[e.Row.RowIndex][0]);
```

Next, we need to get a reference to the "details" HTML anchor in the row. Because `ItemTemplate` were used and the anchor controls in the templates were given IDs, we can accomplish this by using the `FindControl` method of the passed item. If a standard `BoundField` were used instead, the data would have to be accessed using the cells collection (e.g., `e.Row.Cells(1).controls(1)` would access the anchor control in this example). Providing an ID and using `FindControl` eliminates the potential for broken code if the columns are later reordered. The control must be cast to an `HTMLAnchor` because the controls collection is a collection of objects.

After obtaining a reference to the HTML anchor tag, we need to set the `href` property of the anchor to the name of the details page. In addition, the URL needs to include "`BookID=` *n* " where *n* is the ID of the book displayed in the row. The resulting anchor tag in the `GridView` for `BookID = 1` is shown here:

```
<a  href="CH02BookDetailsVB.aspx?BookID=1"
        id="ctl00_PageBody_gvBooks_ctl03_lnkDetails"
   target="_blank">Details</a>
```

The ID is altered by ASP.NET to ensure all server controls have unique IDs. ASP.NET maintains the original and the unique IDs, so the original ID we provided with the `FindControl` method is handled correctly, sparing us from determining the unique ID or dealing with indexing into items and cells.

Example 2-57. GridView with pop-up details window (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02GridViewWithPopupDetailsVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02GridViewWithPopupDetailsVB"
 Title="GridView With Popup Details" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
   GridView With Popup Details (VB)
 </div>
 <asp:GridView ID="gvBooks" Runat="Server"
      AllowPaging="false"
      AllowSorting="false"
      AutoGenerateColumns="false"
      BorderColor="#000080"
      BorderStyle="Solid"
      BorderWidth="2px"
      Caption=""
      HorizontalAlign="Center"
      Width="90%"
      OnRowDataBound="gvBooks_RowDataBound" >
     <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
  <RowStyle cssClass="tableCellNormal" />
  <AlternatingRowStyle cssClass="tableCellAlternating" />
  <SelectedRowStyle CssClass="tableCellSelected" />
  <Columns>

  <asp:BoundField HeaderText="Title"
    DataField="Title"
    ItemStyle-HorizontalAlign="Left" />
  <asp:TemplateField ItemStyle-HorizontalAlign="Center">
    <ItemTemplate>
      <a id="lnkDetails" runat="server"
    target="_blank">Details</a>
    </ItemTemplate>
  </asp:TemplateField>
 </Columns>
  </asp:GridView>
</asp:Content>
```

Example 2-58. GridView with pop-up details window code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02GridViewWithPopupDetailsVB.aspx
 ''' </summary>
 Partial  Class  CH02GridViewWithPopupDetailsVB
  Inherits System.Web.UI.Page

   '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
   Dim dSource As SqlDataSource = Nothing
   Dim dataKeys(0) As String

   If (Not Page.IsPostBack) Then
    'configure the data source to get the data from the database
    dSource = New SqlDataSource( )
    dSource.ConnectionString = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dSource.DataSourceMode = SqlDataSourceMode.DataReader
    dSource.ProviderName = "System.Data.OleDb"
    dSource.SelectCommand = "SELECT BookID, Title " & _
       "FROM Book " & _
       "ORDER BY Title"

    'set the source of the data for the gridview control and bind it
    dataKeys(0) = "BookID"
    gvBooks.DataKeyNames = dataKeys
    gvBooks.DataSource = dSource
    gvBooks.DataBind( )
   End If
  End Sub 'Page_Load

  '''************************************************************************
  ''' <summary>
  ''' This routine is the event handler that is called for each item in the
  ''' GridView after a data bind occurs. It is responsible for setting the
  ''' URL of the anchor tags to the page used to display the details for
  ''' a book
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub gvBooks_RowDataBound(ByVal sender As Object, ByVal e As _
    System.Web.UI.WebControls.GridViewRowEventArgs)
   Const DETAIL_PAGE As String = "CH02BookDetailsVB.aspx"

   Dim bookID As Integer
   Dim anchor As HtmlAnchor

   'check the type of item that was databound and only take action if it
   'was a row in the GridView
   If (e.Row.RowType = DataControlRowType.DataRow) Then
    'get the book ID for the row being bound
    bookID = CInt(gvBooks.DataKeys(e.Row.RowIndex).Item(0))

    'get the anchor tag in the row
    'NOTE: This can be done by using the FindControl method of the passed
    ' item because ItemTemplates were used and the anchor controls in
    ' the templates where given IDs. If a standard BoundField was
    ' used, the data would have to be accessed using the cells
    ' collection (e.g. e.Row.Cells(1).controls(1) would access the
    ' anchor control in this example.
    anchor = CType(e.Row.FindControl("lnkDetails"), _
     HtmlAnchor)

    'set the URL of the anchor tag to the page used to display the book
    'details passing the ID of the book in the querystring
    anchor.HRef = DETAIL_PAGE & "?BookID=" & bookID.ToString( )
   End If
  End Sub 'gvBooks_RowDataBound
 End Class 'CH02GridViewWithPopupDetailsVB
End Namespace
```

Example 2-59. GridView with pop-up details window code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02GridViewWithPopupDetailsCS.aspx
 /// </summary>
 public partial class CH02GridViewWithPopupDetailsCS : System.Web.UI.Page
 {
  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   SqlDataSource dSource = null;
   String[] dataKeys;

   if (!Page.IsPostBack)
   {
    // configure the data source to get the data from the database
    dSource = new SqlDataSource( );
    dSource.ConnectionString = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dSource.DataSourceMode = SqlDataSourceMode.DataReader;
    dSource.ProviderName = "System.Data.OleDb";
    dSource.SelectCommand = "SELECT BookID, Title " +
       "FROM Book " +
       "ORDER BY Title";

    // set the source of the data for the gridview control and bind it
    dataKeys = new string[1] {"BookID"};
    gvBooks.DataKeyNames = dataKeys;
    gvBooks.DataSource = dSource;
    gvBooks.DataBind( );
   }
  } // Page_Load

  ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the GridView's row created
  /// event. It is responsible for setting the icon in the header row to
  /// indicate the current sort column and sort order

  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void gvBooks_RowDataBound(Object sender,
       System.Web.UI.WebControls.GridViewRowEventArgs e)
  {
   const String DETAIL_PAGE = "CH02BookDetailsCS.aspx";

   int bookID;
   HtmlAnchor anchor;

   // check the type of item that was databound and only take action if it
   // was a row in the GridView
   if (e.Row.RowType == DataControlRowType.DataRow)
   {
    // get the book ID for the row being bound
    bookID = (int)(gvBooks.DataKeys[e.Row.RowIndex][0]);

    // 'get the anchor tag in the row
    // NOTE: This can be done by using the FindControl method of the
    // passed item because ItemTemplates were used and the
    // anchor controls in the templates where given IDs. If a
    // standard BoundField was used, the data would have to be
    // accessed using the cells collection (e.g.
    // e.Row.Cells[1].controls[1] would access the anchor control
    // in this example.
    anchor = (HtmlAnchor)(e.Row.FindControl("lnkDetails"));

    // set the URL of the anchor tag to the page used to display the book
    // details passing the ID of the book in the querystring
    anchor.HRef = DETAIL_PAGE + "?BookID=" + bookID.ToString( );
   }
  } //gvBooks_RowDataBound
 } //  CH02GridViewWithPopupDetailsCS
}
```

Example 2-60. Pop-up detail page (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02BookDetailsVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02BookDetailsVB"
 Title="Book Details" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Book Details (VB)
 </div>
 <asp:FormView ID="fvBook" runat="server" HorizontalAlign="Center">
   <ItemTemplate>
  <table width="600" border="0">
    <tr>

   <td rowspan="6" align="center" width="250">
     <img src="images/books/<%#Eval("ImageFilename") %>"alt="Book"></td>
   <td class="labelNormal" width="150">Title: </td>
   <td class="labelNormal" width="325"><%#Eval("Title") %></td>
    </tr>
    <tr>
   <td class="labelNormal">ISBN: </td>
   <td class="labelNormal"><%#Eval("ISBN") %></td>
    </tr>
    <tr>
   <td class="labelNormal">Publisher: </td>
   <td class="labelNormal"><%#Eval("Publisher") %></td>
    </tr>
    <tr>
      <td class="labelNormal">Publish Date: </td>
   <td class="labelNormal"><%#Eval("PublishDate", "{0:MMM yyyy}") %></td>
    </tr>
    <tr>
   <td class="labelNormal">List Price: </td>
   <td class="labelNormal"><%#Eval("ListPrice") %></td>
    </tr>
    <tr>
   <td class="labelNormal">Discounted Price: </td>
   <td class="labelNormal"><%#Eval("DiscountedPrice") %></td>
    </tr>
  </table>
  </ItemTemplate>
   </asp:FormView>
</asp:Content>
```

Example 2-61. Pop-up detail page code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02BookDetailsVB.aspx
 ''' </summary>
 Partial Class CH02BookDetailsVB
  Inherits System.Web.UI.Page

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''

  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
   Dim dSource As SqlDataSource = Nothing
   Dim param As Parameter
   Dim bookID As String

   If (Not Page.IsPostBack) Then
    'get the book ID from the querystring in the URL
    If (IsNothing(Request.QueryString.Item("BookID"))) Then
    'production code needs to handle the page request without the needed
    'information in the querystring here
    Else
    bookID = Request.QueryString.Item("BookID").ToString( )

    'configure the data source to get the data from the database
    dSource = New SqlDataSource( )
    dSource.ConnectionString = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dSource.DataSourceMode = SqlDataSourceMode.DataReader
    dSource.ProviderName = "System.Data.OleDb"
    dSource.SelectCommand = "SELECT Title, ISBN, Publisher, " & _
        "PublishDate, ListPrice, " & _
        "DiscountedPrice, ImageFilename " & _
        "FROM Book " & _
        "WHERE BookID=?"
    param = New Parameter("BookID", TypeCode.Int32, bookID)
    dSource.SelectParameters.Add(param)

    'set the source of the data for the formview control and bind it
    fvBook.DataSource = dSource
    fvBook.DataBind( )
    End If
   End If
  End Sub 'Page_Load
 End Class 'CH02BookDetailsVB
End Namespace
```

Example 2-62. Pop-up detail page code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02BookDetailsCS.aspx

 /// </summary>
 public partial class CH02BookDetailsCS : System.Web.UI.Page
 {
  ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   SqlDataSource dSource = null;
   Parameter param = null;
   String bookID;

   if (!Page.IsPostBack)
   {
    // get the book ID from the querystring in the URL
    if (Request.QueryString["BookID"] == null)
    {
    // production code needs to handle the page request without the
    // needed information in the querystring here
    }
    else
    {
    bookID = Request.QueryString["BookID"].ToString( );
    // configure the data source to get the data from the database
    dSource = new SqlDataSource( );
    dSource.ConnectionString = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dSource.DataSourceMode = SqlDataSourceMode.DataReader;
    dSource.ProviderName = "System.Data.OleDb";
    dSource.SelectCommand = "SELECT Title, ISBN, Publisher, " +
        "PublishDate, ListPrice, " +
        "DiscountedPrice, ImageFilename " +
        "FROM Book " +
        "WHERE BookID=?";
    param = new Parameter("BookID", TypeCode.Int32, bookID);
    dSource.SelectParameters.Add(param);

    // set the source of the data for the gridview control and bind it
    fvBook.DataSource = dSource;
    fvBook.DataBind( );
    }
   }
  } // Page_Load
 } // CH02BookDetailsCS
}
```

Recipe 2.23. Adding a Totals Row to a GridView

Problem

You have a GridView containing numeric information, and you need to display a total of the data in the last row of the grid.

Solution

Enable the output of the footer in the GridView , accumulate the total for the data in the RowDataBound event handler, and then output the total in the GridView footer.

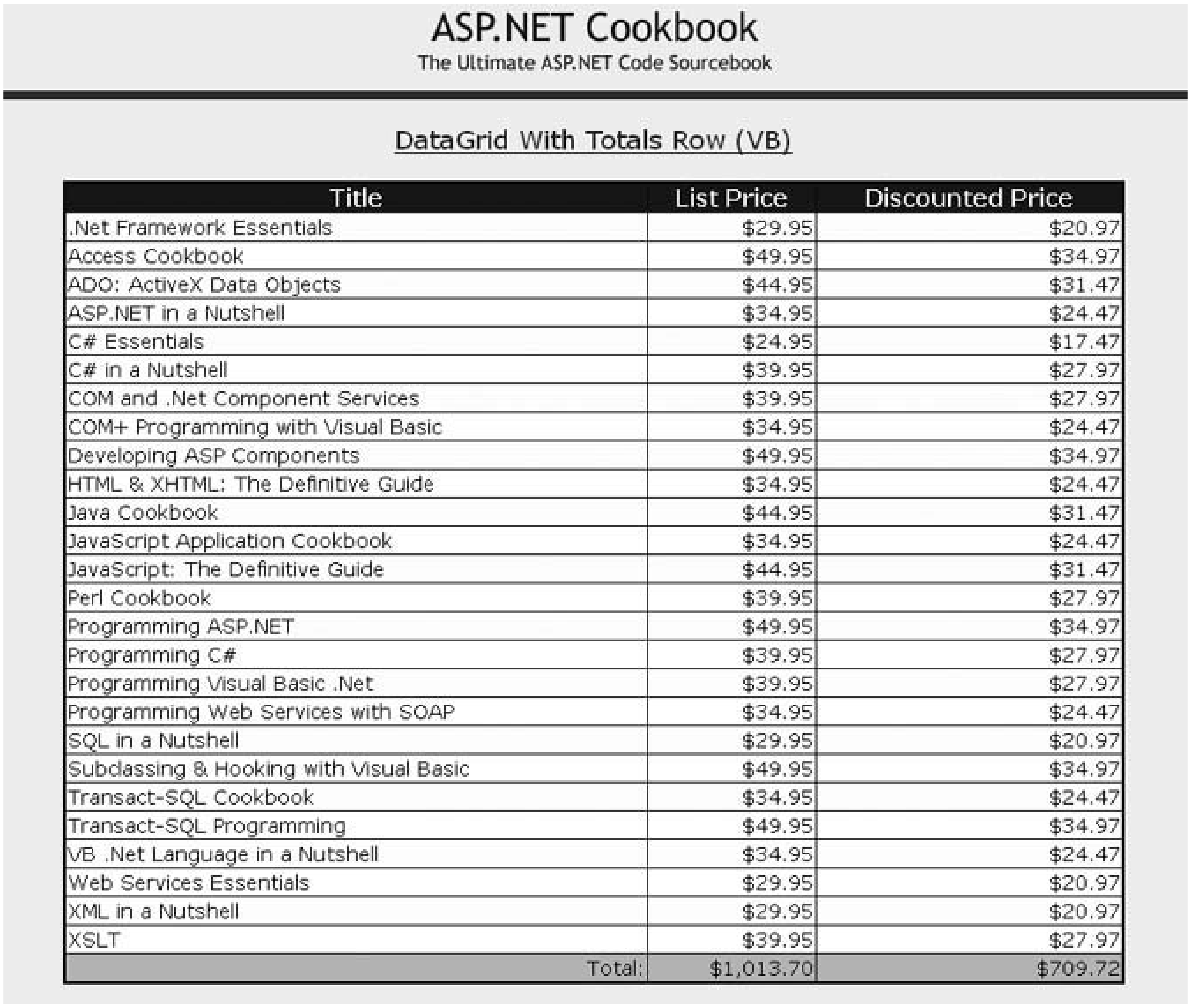
In the *.aspx* file, set the ShowFooter attribute of the asp:GridView element to TRue .

In the code-behind class for the page, use the .NET language of your choice to:

1. Initialize the totals to 0 , and bind the data to the GridView in the normal fashion.
2. In the RowDataBound event handler, add the values for each data row to the accumulated totals.
3. In the RowDataBound event handler, set the total values in the footer when the footer is data bound.

Figure 2-25 shows some typical output. Examples 2-63, 2-64 through 2-65 show the *.aspx* file and code-behind files for an application that produces this output.

Figure 2-25. GridView with totals row output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

DataGrid With Totals Row (VB)

Title	List Price	Discounted Price
.Net Framework Essentials	$29.95	$20.97
Access Cookbook	$49.95	$34.97
ADO: ActiveX Data Objects	$44.95	$31.47
ASP.NET in a Nutshell	$34.95	$24.47
C# Essentials	$24.95	$17.47
C# in a Nutshell	$39.95	$27.97
COM and .Net Component Services	$39.95	$27.97
COM+ Programming with Visual Basic	$34.95	$24.47
Developing ASP Components	$49.95	$34.97
HTML & XHTML: The Definitive Guide	$34.95	$24.47
Java Cookbook	$44.95	$31.47
JavaScript Application Cookbook	$34.95	$24.47
JavaScript: The Definitive Guide	$44.95	$31.47
Perl Cookbook	$39.95	$27.97
Programming ASP.NET	$49.95	$34.97
Programming C#	$39.95	$27.97
Programming Visual Basic .Net	$39.95	$27.97
Programming Web Services with SOAP	$34.95	$24.47
SQL in a Nutshell	$29.95	$20.97
Subclassing & Hooking with Visual Basic	$49.95	$34.97
Transact-SQL Cookbook	$34.95	$24.47
Transact-SQL Programming	$49.95	$34.97
VB .Net Language in a Nutshell	$34.95	$24.47
Web Services Essentials	$29.95	$20.97
XML in a Nutshell	$29.95	$20.97
XSLT	$39.95	$27.97
Total:	$1,013.70	$709.72

Discussion

The best way to describe the addition of a totals row to a `GridView` is by example. In this recipe, you'll want to create the `GridView` differently than normal. In the `asp:GridView` element, set the `ShowFooter` attribute to `true` to cause a footer to be output when the control is rendered. Then, you place the totals data in the footer.

```
<asp:GridView ID="gvBooks" Runat="Server"
   AllowPaging="false"
   AllowSorting="false"
   AutoGenerateColumns="false"
    BorderColor="#000080"
   BorderStyle="Solid"
    BorderWidth="2px"
   Caption=""
   HorizontalAlign="Center"
```

```
    ShowFooter="true"
     Width="90%"
    OnRowDataBound="gvBooks_RowDataBound" >
```

Next, add a `FooterStyle` element to format all of the columns in the footer with a stylesheet class and horizontal alignment:

```
 <FooterStyle CssClass="tableCellSelected" HorizontalAlign="Right"/>
```

All columns are defined in the `Columns` element as `asp:TemplateField` columns. This provides a lot of flexibility in the display of the columns. The first column contains only an `ItemTemplate` that is bound to the Title field in the `DataSource` . The `FooterText` property of this column is set to "`Total` :" to display the label for the other values in the footer.

```
 <asp:TemplateField HeaderText="Title" FooterText="Total:">
   <ItemTemplate>
  <%# Eval("Title") %>
   </ItemTemplate>
 </asp:TemplateField>
```

The second and third columns contain an `ItemTemplate` element to define the format of the data placed in the rows of the grid and a `FooterTemplate` element to define the format of the data placed in the footer of the respective columns:

```
 <asp:TemplateField HeaderText="List Price"
      ItemStyle-HorizontalAlign="Right">
  <ItemTemplate>
    <asp:Literal id="lblListPrice" runat="server"
    text='<%# Eval("ListPrice") %>' />
  </ItemTemplate>
  <FooterTemplate>
   <asp:Literal id="lblListPriceTotal" runat="server" />
  </FooterTemplate>
 </asp:TemplateField>
```

In the code-behind, two private variables (`mListPriceTotal` and `mDiscountedPriceTotal`) are declared at the class level to store the accumulated sum for each of the price columns. The `Page_Load` method is identical to previous recipes, except for the addition of the code to set `mListPriceTotal` and `mDiscountedPriceTotal` to zero before the data binding is performed.

The `RowDataBound` event is used to accumulate the sum of the prices as the rows in the `GridView` are bound. You can do this because the data binding always starts at the top of the grid and ends at the bottom. Because the `RowDataBound` event method is called for every row in the grid, you must determine what row this event applies to by checking the `ItemType` of the passed event arguments. Several groups of item types are needed here, so a `Select Case` statement (`switch` in C#) is used.

When the item type is a data row, you need to get the values in the list price and discounted price columns and add them to the appropriate total variables. Getting the price values requires getting the price values from the data passed to the method (`e.Row. DataItem`), adding the price data to the totals, getting a reference to the controls used to display the data, and setting the price value in the controls for the row. Getting a reference to the control is the trickiest part. The easiest and most flexible approach is to use `Literal` controls in the `ItemTemplates` of the `GridView` defined in the *.aspx* file. By setting the IDs of the literal controls, the `FindControl` method of the row being data bound can be used to get a reference to the desired control.

> If the IDs of the controls in the `ItemTemplates` are undefined, the only way to get a reference to a control is to index into the cells and controls collections of the row. In this example, the list price control is in the second column of the grid. Cells in a `GridView` are created with a literal control before and after the controls you define in a column; therefore, the list price control is the second control in the controls collection of the cell. Getting a reference to the list price control using this method would be done with `listPriceControl = e.Row.Cells(1). controls(1)` . This approach depends on column layout: rearranging columns would break code that uses this approach. The `FindControl` method is easier to maintain and less likely to be broken by changing the user interface.
>
> Literal controls are used in this example because they are rendered without the addition of other controls and because accessing the price value is as simple as getting the value of the text property of the control. An `asp:Label` control would seem like a good option here; however, it is created as three literal controls in the `GridView` , making it necessary to index into the controls collection of the control returned by the `FindControl` method to get the needed price value.

When the item is the footer, all data rows have been processed, and you have the totals for the price columns in the `mListPriceTotal` and `mDiscountedPriceTotal` variables. Now you need to output these totals in the controls placed in the footer. This is done by using the `FindControl` method of the passed item to get a reference to the controls in the footer. After a reference to the control is obtained, the text property is set to the total for the column. In our example, the totals are being formatted to be displayed in currency format with two decimal places.

Example 2-63. GridView with totals row (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH02GridViewWithTotalsRowVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH02GridViewWithTotalsRowVB"
 Title="GridView With Totals Row" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
```

```
<div align="center" class="pageHeading">
 GridView With Totals Row (VB)
</div>
<asp:GridView ID="gvBooks" Runat="Server"
     AllowPaging="false"
     AllowSorting="false"
     AutoGenerateColumns="false"
     BorderColor="#000080"
     BorderStyle="Solid"
     BorderWidth="2px"
     Caption=""
     HorizontalAlign="Center"
     ShowFooter="true"
     Width="90%"
     OnRowDataBound="gvBooks_RowDataBound" >
 <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
 <RowStyle cssClass="tableCellNormal" />
 <AlternatingRowStyle cssClass="tableCellAlternating" />
 <FooterStyle CssClass="tableCellSelected" HorizontalAlign="Right"/>
 <Columns>
  <asp:TemplateField HeaderText="Title" FooterText="Total:">
   <ItemTemplate>
   <%# Eval("Title") %>
   </ItemTemplate>
  </asp:TemplateField>

  <asp:TemplateField HeaderText="List Price"
    ItemStyle-HorizontalAlign="Right">
   <ItemTemplate>
   <asp:Literal id="lblListPrice" runat="server"
     text='<%# Eval("ListPrice") %>' />
   </ItemTemplate>
   <FooterTemplate>
   <asp:Literal id="lblListPriceTotal" runat="server" />
   </FooterTemplate>
  </asp:TemplateField>

  <asp:TemplateField HeaderText="Discounted Price"
      ItemStyle-HorizontalAlign="Right">
   <ItemTemplate>

   <asp:Literal id="lblDiscountedPrice" runat="server"
    text='<%# Eval("DiscountedPrice") %>' />
   </asp:Label>
  </ItemTemplate>
  <FooterTemplate>
   <asp:Literal id="lblTotalDiscountedPrice"
    runat="server" />
  </FooterTemplate>
 </asp:TemplateField>
</Columns>
 </asp:GridView>
```

```
</asp:Content>
```

Example 2-64. GridView with totals row code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH02GridViewWithTotalsRowVB.aspx
 ''' </summary>
 Partial  Class  CH02GridViewWithTotalsRowVB
  Inherits System.Web.UI.Page

  'variables used to accumulate the sum of the prices
  Private mListPriceTotal As Decimal
  Private mDiscountedPriceTotal As Decimal
  '''*******************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
     Dim dSource As SqlDataSource = Nothing

     If (Not Page.IsPostBack) Then
       'configure the data source to get the data from the database
    dSource = New SqlDataSource( )
    dSource.ConnectionString = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
   dSource.DataSourceMode = SqlDataSourceMode.DataSet

   dSource.ProviderName = "System.Data.OleDb"
   dSource.SelectCommand = "SELECT Title, ListPrice, DiscountedPrice " & _
      "FROM Book " & _
      "ORDER BY Title"

   'set total values to 0 before data binding
   mListPriceTotal = 0
   mDiscountedPriceTotal = 0

   'set the source of the data for the gridview control and bind it
   gvBooks.DataSource = dSource
   gvBooks.DataBind( )
    End If
  End Sub 'Page_Load

  '''***********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the GridView's row data
  ''' bound event. It is responsible for formatting the data in the
  ''' columns of the GridView
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub gvBooks_RowDataBound(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs)
   Dim rowData As DataRowView
   Dim price As Decimal
   Dim listPriceLabel As System.Web.UI.WebControls.Literal
   Dim discountedPriceLabel As System.Web.UI.WebControls.Literal
   Dim totalLabel As System.Web.UI.WebControls.Literal

   'check the type of item that was databound and only take action if it
   'was a row in the gridview
   Select Case (e.Row.RowType)
    Case DataControlRowType.DataRow
    'get the data for the item being bound
    rowData = CType(e.Row.DataItem, _
      DataRowView)

    'get the value for the list price and add it to the sum
    price = CDec(rowData.Item("ListPrice"))
    mListPriceTotal += price

    'get the control used to display the list price
    'NOTE: This can be done by using the FindControl method of the
    ' passed item because ItemTemplates were used and the anchor
    ' controls in the templates where given IDs. If a standard
    ' BoundField was used, the data would have to be accessed
    ' using the cellscollection (e.g. e.Row.Cells(1).controls(1)
    ' would access the label control in this example.
    listPriceLabel = CType(e.Row.FindControl("lblListPrice"), _
       System.Web.UI.WebControls.Literal)

    'now format the list price in currency format
    listPriceLabel.Text = price.ToString("C2")

    'get the value for the discounted price and add it to the sum
```

```
    price = CDec(rowData.Item("DiscountedPrice"))
    mDiscountedPriceTotal += price

    'get the control used to display the discounted price
    discountedPriceLabel = CType(e.Row.FindControl("lblDiscountedPrice"), _
          System.Web.UI.WebControls.Literal)

    'now format the discounted price in currency format
    discountedPriceLabel.Text = price.ToString("C2")

    Case DataControlRowType.Footer
    'get the control used to display the total of the list prices
    'and set its value to the total of the list prices
    totalLabel = CType(e.Row.FindControl("lblListPriceTotal"), _
      System.Web.UI.WebControls.Literal)
    totalLabel.Text = mListPriceTotal.ToString("C2")

    'get the control used to display the total of the discounted prices
    'and set its value to the total of the discounted prices
    totalLabel = CType(e.Row.FindControl("lblTotalDiscountedPrice"), _
       System.Web.UI.WebControls.Literal)
    totalLabel.Text = mDiscountedPriceTotal.ToString("C2")
    Case Else
    'DataControlRowType.EmptyDataRow, DataControlRowType.Header,
    'DataControlRowType.Pager, or DataControlRowType.Separator
    'no action required
   End Select
  End Sub 'gvBooks_RowDataBound
 End Class 'CH02GridViewWithTotalsRowVB
End Namespace
```

Example 2-65. GridView with totals row code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.Common;
using System.Data.OleDb;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH02GridViewWithTotalsRowCS.aspx
 /// </summary>
 public partial class CH02GridViewWithTotalsRowCS : System.Web.UI.Page
```

```
{
 // variables used to accumulate the sum of the prices
 private Decimal mListPriceTotal;
 private Decimal mDiscountedPriceTotal;

 ///***********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
  SqlDataSource dSource = null;

  if (!Page.IsPostBack)
  {
   // configure the data source to get the data from the database
   dSource = new SqlDataSource( );
   dSource.ConnectionString = ConfigurationManager.
   ConnectionStrings["dbConnectionString"].ConnectionString;
   dSource.DataSourceMode = SqlDataSourceMode.DataReader;
   dSource.ProviderName = "System.Data.OleDb";
   dSource.SelectCommand = "SELECT Title, ListPrice, DiscountedPrice " +
      "FROM Book " +
      "ORDER BY Title";

   // set total values to 0 before data binding
   mListPriceTotal = 0;
   mDiscountedPriceTotal = 0;

   // set the source of the data for the gridview control and bind it
   gvBooks.DataSource = dSource;
   gvBooks.DataBind( );
  }
 } // Page_Load

 ///***********************************************************************
 /// <summary>
 /// This routine provides the event handler for the GridView's row created
 /// event. It is responsible for setting the icon in the header row to
 /// indicate the current sort column and sort order
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void gvBooks_RowDataBound(Object sender,
     System.Web.UI.WebControls.GridViewRowEventArgs e)
 {
  DbDataRecord rowData;
```

```
Decimal price;
System.Web.UI.WebControls.Literal listPriceLabel;
System.Web.UI.WebControls.Literal discountedPriceLabel;
System.Web.UI.WebControls.Literal totalLabel;

// check the type of item that was databound and only take action if it
// was a row in the gridview
switch (e.Row.RowType)
{
 case DataControlRowType.DataRow:
 // get the data for the item being bound
 rowData = (DbDataRecord)(e.Row.DataItem);

 // get the value for the list price and add it to the sum
 price = (Decimal)(rowData["ListPrice"]);
 mListPriceTotal += price;

 // get the control used to display the list price
 // NOTE: This can be done by using the FindControl method of the
 // passed item because ItemTemplates were used and the anchor
 // controls in the templates where given IDs. If a standard
 // BoundField was used, the data would have to be accessed
 // using the cells collection (e.g. e.Row.Cells(1).controls(1)
 // would access the label control in this example.
 listPriceLabel = (System.Web.UI.WebControls.Literal)
   (e.Row.FindControl("lblListPrice"));

 // now format the list price in currency format
 listPriceLabel.Text = price.ToString("C2");

 // get the value for the discounted price and add it to the sum
 price = (Decimal)(rowData["DiscountedPrice"]);
 mDiscountedPriceTotal += price;

 // get the control used to display the discounted price
 discountedPriceLabel = (System.Web.UI.WebControls.Literal)
     (e.Row.FindControl("lblDiscountedPrice"));

 // now format the discounted price in currency format
 discountedPriceLabel.Text = price.ToString("C2");
 break;

 case DataControlRowType.Footer:
 // get the control used to display the total of the list prices
 // and set its value to the total of the list prices
 totalLabel = (System.Web.UI.WebControls.Literal)
  (e.Row.FindControl("lblListPriceTotal"));
 totalLabel.Text = mListPriceTotal.ToString("C2");

 // get the control used to display the total of the discounted prices
 // and set its value to the total of the discounted prices
```

```
    totalLabel = (System.Web.UI.WebControls.Literal)

     (e.Row.FindControl("lblTotalDiscountedPrice"));
    totalLabel.Text = mDiscountedPriceTotal.ToString("C2");
    break;

    default:
    // DataControlRowType.EmptyDataRow, DataControlRowType.Header,
    // DataControlRowType.Pager, or DataControlRowType.Separator
    // no action required
    break;
   } // switch (e.Row.RowType)
  } //gvBooks_RowDataBound
 } // CH02GridViewWithTotalsRowCS
}
```

PREV

PREV

Chapter 3. Validation

PREV

PREV

3.0 Introduction

ASP.NET validation controls (also known as *validators*) simplify the task of ensuring that data is entered correctly on forms. For most validations, no code is required in the *.aspx* file or the code-behind class. You add a validator to the *.aspx* file, have it reference an input control (a server control) elsewhere on the page, and set one or more of its validation attributes (such as `MinimumValue` or `MaximumValue`, which specify the minimum and maximum values of a validation range). ASP.NET does all the rest. You can combine validators to provide multiple validations on a single input, such as a `RequiredFieldValidator` and a `RangeValidator`, which perform as their names imply.

Validation can be performed on the client and server. By default, validators perform their validation automatically on `postback` in server code. However, if the user has a browser that supports DHTML and client-side validation is enabled, validators can perform their validation using client script. Client-side validation is handy whenever you want to avoid a round trip to the server for server-side validation, such as when you want to make sure an entry is provided in a text box. Regardless of whether client-side validation is performed, server-side validation is always a good idea if only to ensure that validation takes place, even when the user's browser doesn't support DHTML.

This chapter includes a useful collection of recipes for validating data, starting with automatic, attribute-oriented validation, next dealing with custom validation, and ending with validation groups. When you perform custom validation, you intercept an input control's validation call and provide you own validation logic (by adding your own custom JavaScript and server-side code). Custom validatior is the focus of two of the chapter's recipes, which show you how to require a user to make a selectio from a drop-down list and how to require valid user input data, such as a password that matches an entry in a database.

Validation groups are new to ASP.NET 2.0 and were introduced to support the concept of multiple logical sections on a form. A common scenario is when you have two logical sections on a form, one for supporting login and the other for new user registration, both with their own Submit buttons, sucl as that shown in Figure 3-14. Handling this scenario in ASP.NET 1.x was difficult because clicking either button would invoke *all* the validators for the page, which was not generally intended. With ASP.NET 2.0, however, you can group controls for purposes of validation and associate them with the button the user clicks. That is, you can have one group of controls validated when a Login button is clicked and another when a Register button is clicked. The last two recipes show you how to handle this scenario easily using validation groups. The first shows the basics and the second goes into more depth by showing you how to handle the validation under programmatic control, which is useful wher you want to perform your own nonstandard validation, such as when you want to check a new user's registration against a database.

All validators, except the `RequiredFieldValidator`, allow the control being validated to be left blank. This subtle point is worth noting, as you may need to account for this behavior in your code when using ASP.NET's automatic validation.

PREV

PREV

Recipe 3.2. Requiring That Data Be Entered in a Field

Problem

You need to ensure that a user has entered data in a text box, such as a first or last name on a registration form.

Solution

Add a `RequiredFieldValidator` control to the *.aspx* file, and use the event handler of the control that completes the user's entry for the page to verify that validation was successful.

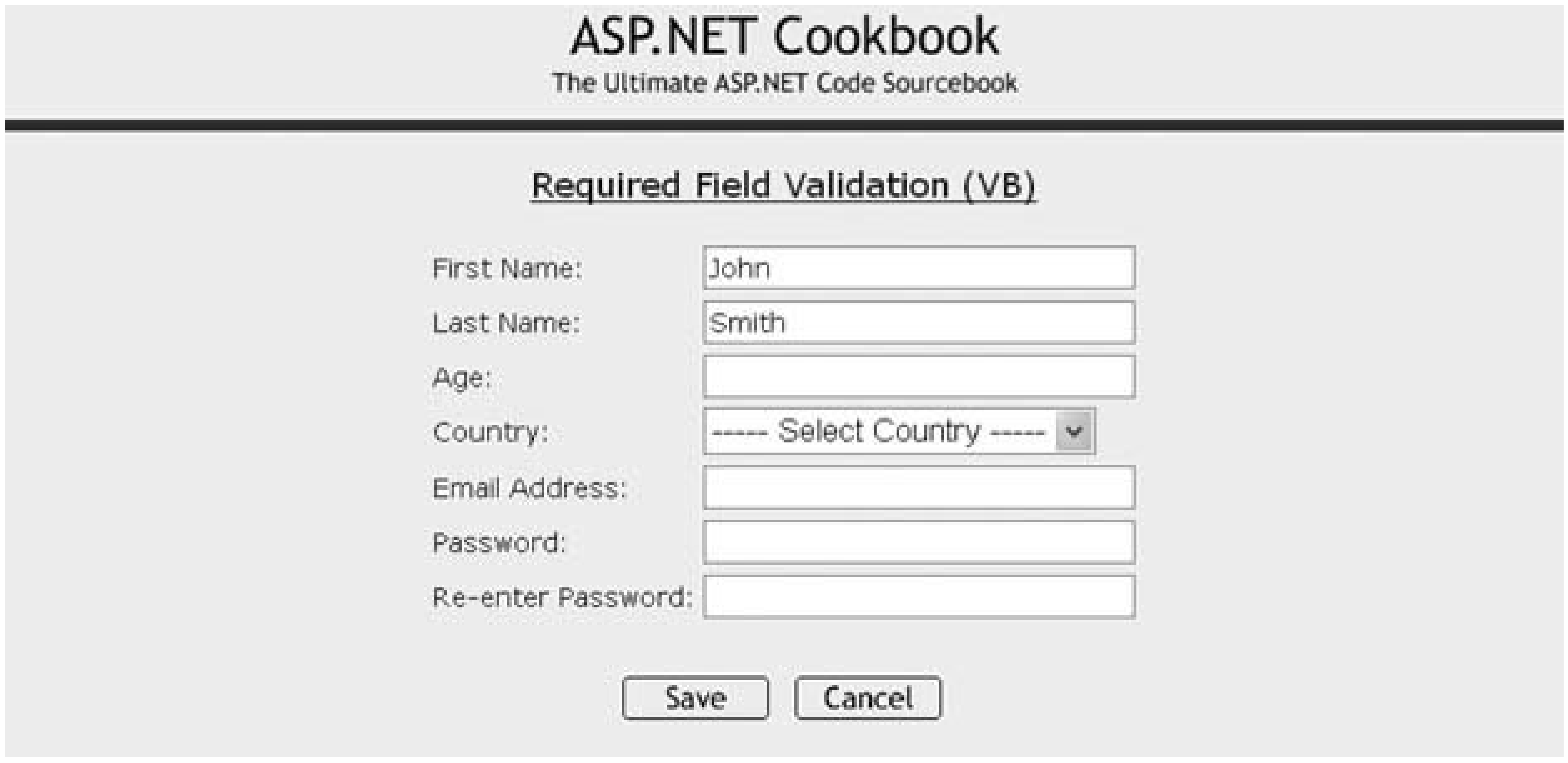
In the *.aspx* file:

1. Add a `RequiredFieldValidator` control for each text box in which data must be entered.
2. Set the `ControlToValidate` attribute to the ID of the control to validate.
3. Add Save and Cancel (or equivalently named) buttons.
4. Set the Save button's `CausesValidation` attribute to `true` to have validation performed when the button is clicked (set it to `False` for the Cancel button).

In the code-behind class, use the .NET language of your choice to add code to the event handler for the Save button's click event that checks the `Page.IsValid` property and verifies that all validation was successful.

Figure 3-1 shows a typical user input form with fields for First Name and Last Name and several othe types of information. Figure 3-2 shows the same form with validation error messages that appear when the user fails to complete the First Name and Last Name fields. Example 3-1 shows the *.aspx* file that implements the form, and Examples 3-2 and 3-3 show the VB and C# code-behind files needed to complete the application.

Figure 3-1. Form with required field validation outputnormal



ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

Required Field Validation (VB)

First Name:	John
Last Name:	Smith
Age:	
Country:	----- Select Country -----
Email Address:	
Password:	
Re-enter Password:	

Figure 3-2. Form with required field validation outputwith error message

Discussion

When you need to insist that a user enter data into a text box, a common requirement for forms used to register new users for a site or service, the `RequiredFieldValidator` control provides a straightforward way to enforce the rule. We've used the control to require completion of the First Name and Last Name text boxes of a simple registration form (see Figures 3-1 and Figures 3-2). You need to assign a `RequiredFieldValidator` control to each text box you wish to check. Each validator control must be placed on the form at the exact spot where you want its error message to be displayed (typically just after the text box it validates), and the`ControlToValidate` attribute of the validator must be set to the ID of the text box as shown in the following code snippet. In our example, the names of the First Name and Last Name text boxes are`txtFirstName` and `txtLastName`, respectively.

```
<asp:RequiredFieldValidator id="rfvFirstName"
  Runat="server"
  ControlToValidate="txtFirstName"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True">
   <img src="images/arrow_alert.gif" alt="arrow"/>
   First Name Is Required
</asp:RequiredFieldValidator>
```

The `Display` attribute must be set to `Dynamic, Static`, or `None. Dynamic` causes ASP.NET to output the HTML related to the validator error message only when an error message is to be output.`Static` causes HTML related to the validator error message to be output at all times even when an error message is not output. `None` prevents any HTML related to the validator error message from being output; this setting is useful when you plan to use an error summary and do not wish to display an error message at the specific field. (See Recipe 3.5 for an example that uses an error summary.) In our example, the `Display` attribute is set to `Dynamic` so an error message is issued only when validation fails:

```
<asp:RequiredFieldValidator id="rfvFirstName"
  Runat="server"
  ControlToValidate="txtFirstName"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True">
   <img src="images/arrow_alert.gif" alt="arrow"/>
   First Name Is Required
</asp:RequiredFieldValidator>
```

The `EnableClientScript` attribute can be set to `true` or `False` as a function of how you want validation performed. Setting the attribute to`TRue` causes validation to be performed on the client and on the server when the form is submitted. Setting the attribute to`False` causes validation to be performed only on the server when the form is submitted. See Recipe 3.6 for an example showing when you may want to set this attribute to`False`. In our example, we have set the `EnableClientScript` attribute to `true` so validation is performed on the client and the server:

```
<asp:RequiredFieldValidator id="rfvFirstName"
  Runat="server"
```

```
      ControlToValidate="txtFirstName"
   CssClass="AlertText"
   Display="Dynamic"
   EnableClientScript="True">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    First Name Is Required
</asp:RequiredFieldValidator>
```

The error message to be output when validation fails is placed between the open and close tags of the control. The message can include HTML, as shown here, where an HTML image tag comes first, followed by text:

```
<asp:RequiredFieldValidator id="rfvFirstName"
   Runat="server"
   ControlToValidate="txtFirstName"
   CssClass="AlertText"
   Display="Dynamic"
   EnableClientScript="True">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    First Name Is Required
</asp:RequiredFieldValidator>
```

In our application, two buttons are provided on the form to allow the user to submit or cancel the page. The Save button causes the form to be submitted and the data the user has entered to be validated, while the Cancel button causes validation to be bypassed. Validation is requested by setting the `CausesValidation` attribute to `true` for the Save button and `False` for the Cancel button:

```
<input id="btnSave" runat="server" type="button"
     value="Save" causesvalidation="true"
     onserverclick="btnSave_ServerClick"/>
<input id="btnCancel" runat="server" type="button"
     value="Cancel" causesvalidation="false"/>
```

With the setup done in the *.aspx* file, the code-behind requires a simple check of the`Page.IsValid` property in the event handler for the Save button's click event. This is done to ensure all client-and server-side validation was successful before processing the form data.

See Also

Recipes 3.5 and 3.6

Example 3-1. Form with required field validation (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
      CodeFile="CH03RequiredFieldValidationVB.aspx.vb"
       Inherits="ASPNetCookbook.VBExamples.CH03RequiredFieldValidationVB"
     title="Validator - Required Field" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
   <div align="center" class="pageHeading">
     Required Field Validation (VB)
   </div>
   <table align="center" class="dataEntry"></tr>
  <td class="LabelText">First Name: </td>
        <td>
          <asp:TextBox id="txtFirstName" Runat="server"
                      Columns="30" CssClass="LabelText" />
    <asp:RequiredFieldValidator id="rfvFirstName"
      Runat="server"
      ControlToValidate="txtFirstName"
      CssClass="alertText"
      Display="Dynamic"
      EnableClientScript="True">
       <img src="images/arrow_alert.gif" alt="arrow"/>
    First Name Is Required
    </asp:RequiredFieldValidator>
     </td>
     </tr>
     </tr>
  <td class="LabelText">Last Name: </td>
  <td>
    <asp:TextBox id="txtLastName" Runat="server"
      Columns="30" CssClass="LabelText" />
    <asp:RequiredFieldValidator id="rfvLastName"
      Runat="server"
      ControlToValidate="txtLastName"
      CssClass="alertText"
      Display="Dynamic"
      EnableClientScript="True">
   <img src="images/arrow_alert.gif" alt="arrow"/>
   Last Name Is Required
      </asp:RequiredFieldValidator>
    </td>
     </tr>
  </tr>
    <td class="LabelText">Age: </td>
    <td>
     <asp:TextBox id="txtAge" Runat="server"
      Columns="30" CssClass="LabelText" />
    </td>
     </tr>
     </tr>
    <td class="LabelText">Country: </td>
    <td>
    <asp:DropDownList id="ddCountry" Runat="server" >
         <asp:ListItem Selected="True"
     Value="0">----- Select Country -----</asp:ListItem>

       <asp:ListItem Value="1">Canada</asp:ListItem>
       <asp:ListItem Value="2">United States</asp:ListItem>
       </asp:DropDownList>
       </td>
     </tr>
     </tr>
     <td class="LabelText">Email Address: </td>
     <td>
      <asp:TextBox id="txtEmailAddress" Runat="server"
     Columns="30" CssClass="LabelText" />
   </td>
 </tr>
 </tr>
      <td class="LabelText">Password: </td>
      <td>
       <asp:TextBox id="txtPassword1" Runat="server"
    TextMode="Password"
     Columns="30" CssClass="LabelText" />
   </td>
    </tr>
    </tr>
  <td class="LabelText">Re-enter Password: </td>
  <td>
       <asp:TextBox id="txtPassword2" Runat="server"
    TextMode="Password"
     Columns="30" CssClass="LabelText" />
   </td>
    </tr>
    </tr>
  <td colspan="2">
      <br/>
       <table align="center" width="50%"></tr>
      <td align="center">
      <input id="btnSave" runat="server" type="button"
    value="Save" causesvalidation="true"
    onserverclick="btnSave_ServerClick"/>
       </td>
       <td align="center">
          <input id="btnCancel" runat="server" type="button"
    value="Cancel" causesvalidation="false"
    onserverclick="btnCancel_ServerClick" />
        </td>
      </tr>
   </table>
  </td>
 </tr>
  </table>
</asp:Content>
```

Example 3-2. Form with required field validation code-behind (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for CH03RequiredFieldValidationVB.aspx
  ''' </summary>
  Partial Class CH03RequiredFieldValidationVB
 Inherits System.Web.UI.Page
 '''*********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the save button click
 ''' event. It is responsible for processing the form data.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnSave_ServerClick(ByVal sender As Object, _
         ByVal e As System.EventArgs)
    If (Page.IsValid) Then
   'process form data and save as required for application
    End If
 End Sub 'btnSave_ServerClick
 '''*********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the cancel button click
 ''' event. It is responsible for processing the form data.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnCancel_ServerClick(ByVal sender As Object, _
        ByVal e As System.EventArgs)
   'perform any cancel operations required for application
 End Sub 'btnCancel_ServerClick
  End Class 'CH03RequiredFieldValidationVB
End Namespace
```

Example 3-3. Form with required field validation code-behind (.cs)

```
using System;
namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
   ///  CH03RequiredFieldValidationCS.aspx
  /// </summary>
   public partial class CH03RequiredFieldValidationCS : System.Web.UI.Page
  {
 ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the save button click
 /// event. It is responsible for processing the form data.
 /// </summary>

 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnSave_ServerClick(Object sender,
          System.EventArgs e)
 {
   if (Page.IsValid)
   {
  // process form data and save as required for application
   }
 } //btnSave_ServerClick
 ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the cancel button click
 /// event. It is responsible for processing the form data.
 /// </summary>
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnCancel_ServerClick(Object sender,
         System.EventArgs e)
 {
   // perform any cancel operations required for application
 }  // btnCancel_ServerClick
  } // CH03RequiredFieldValidationCS
}
```

PREV

Recipe 3.3. Requiring Data to Be in a Range

Problem

You need to ensure data entered by a user is within a defined rangefor example, between two numbers, currency values, dates, or alphabetic characters.

Solution

Add a `RangeValidator` control to the *.aspx* file for each `TextBox` control to be checked, set the minimum and maximum acceptable values for the range, and verify that validation was successful from within the event handler of the control that completes the user's entry for the page.

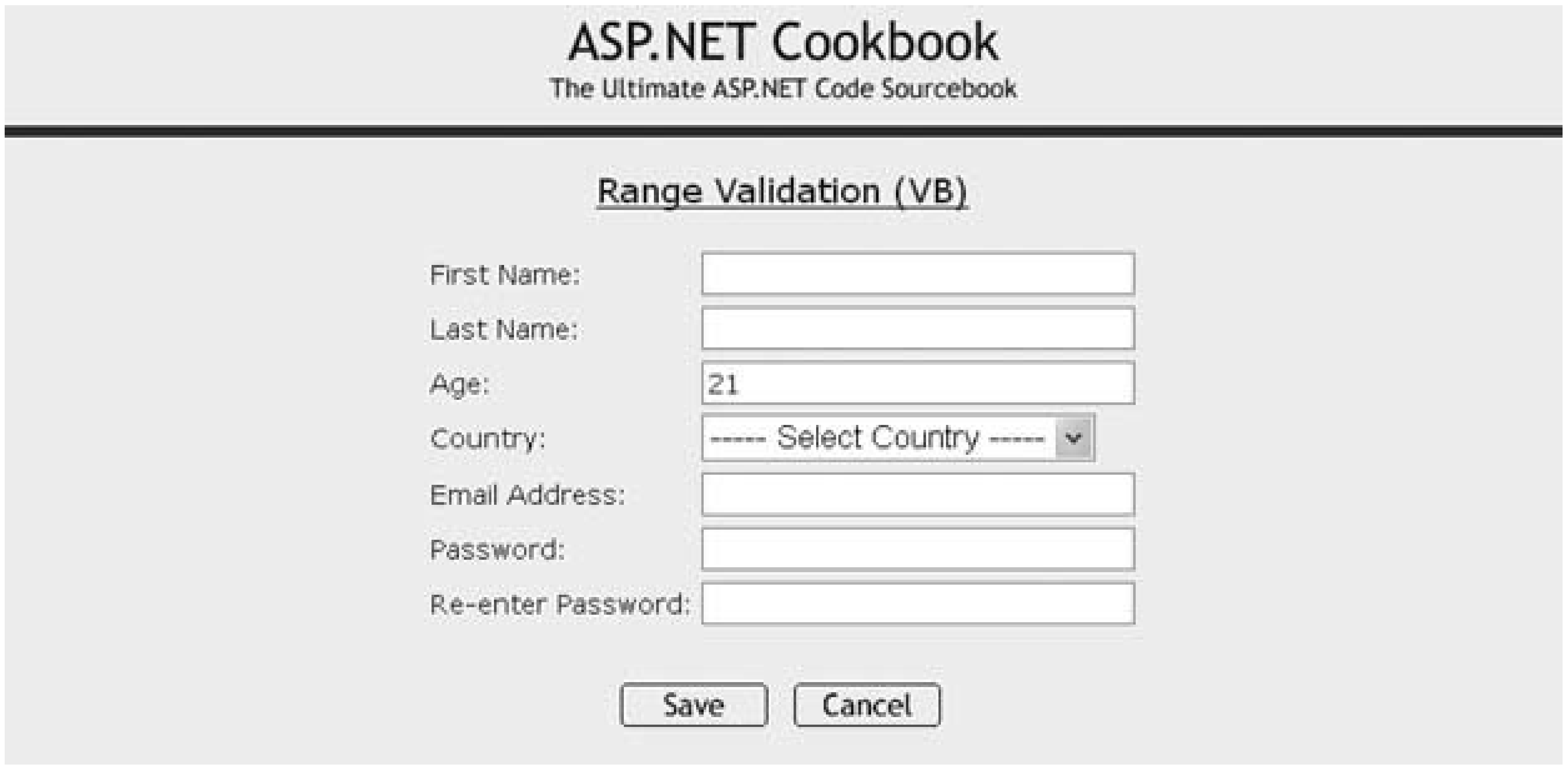
In the *.aspx* file:

1. Add a `RangeValidator` control for each text box in which the user must enter data within a specified range.
2. Set the `ControlToValidate` attribute to the ID of the control to validate.
3. Set the control's `MinimumValue` and `MaximumValue` attributes to the minimum and maximum values for the valid range.
4. Add Save and Cancel (or equivalently named) buttons.
5. Set the Save button's `CausesValidation` attribute to `TRue` to have validation performed when the button is clicked (set it to `False` for the Cancel button).

In the code-behind class for the page, use the .NET language of your choice to add code to the event handler for the Save button's click event to check the `Page.IsValid` property and verify that all validation was successful. (See Recipe 3.1 for details.)

Figure 3-3 shows the user input form introduced in Recipe 3.1 with normal, error-free output. Figure 3-4 shows the same form with the error message that appears on the form when the data entered into the Age field falls outside a predeterminedrange. Example 3-4 shows the .aspx file that implements the form, and Examples 3-2 and 3-3 (see Recipe 3.1) show the companion code-behind files.

Figure 3-3. Form with range validation outputnormal



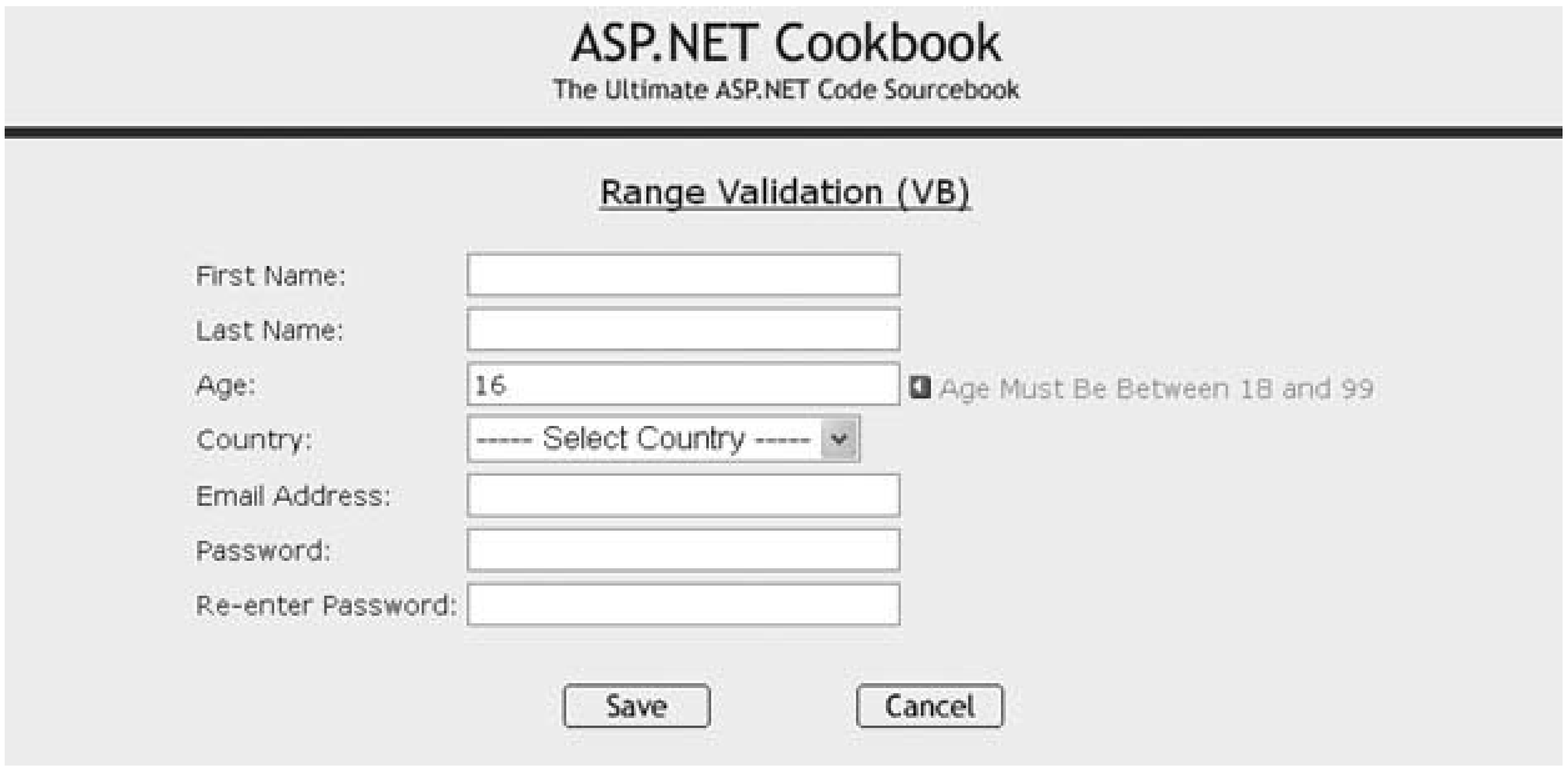
Range Validation (VB)

First Name:	
Last Name:	
Age:	21
Country:	----- Select Country -----
Email Address:	
Password:	
Re-enter Password:	

Discussion

To make sure a user enters data in a text box within a defined range, place a`RangeValidator` control on the form and assign it the text box to be validated. To create the form shown inFigures 3-3 and Figures 3-4, for example, we added an `asp:RangeValidator` control to the *.aspx* file that implements the form and assigned it to the Age text box to ensure the entered data is within the range 18 to 99. You must place the validator on the form at the exact location where you want the control's error message to be displayed, which in our case is just to the right of the Age text box.

Figure 3-4. Form with range validation outputwith error message



To assign a `RangeValidator` control to a text box or other control type, you must set its `ControlToValidate` attribute to the ID of the control you wish to validate. In our example, the ID of the Age text box is `txtAge`:

```
<asp:RangeValidator id="rvAge" Runat="server"
  ControlToValidate="txtAge"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"
   MinimumValue="18"
   MaximumValue="99"
  Type="Integer">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    Age Must Be Between 18 and 99
</asp:RangeValidator>
```

To specify a valid range, you must set the `MinimumValue` and `MaximumValue` attributes of the `RangeValidator` control. In our example, we have set the lowest acceptable value to `18` and the highest to `99`. These values are inclusive, which means that ages 18 and 99 are also acceptable.

```
<asp:RangeValidator id="rvAge" Runat="server"
  ControlToValidate="txtAge"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"
  MinimumValue="18"
  MaximumValue="99"
```

```
   Type="Integer">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    Age Must Be Between 18 and 99
    </asp:RangeValidator>
```

Our example focuses on using a range of two numbers, but you can use a range of dates, times, currency values, alphabetic characters, etc.

To change the minimum and maximum values dynamically, you can set them in the code-behind, as shown next. You might want to do this, for example, when determining the range on the fly.

VB

```
Dim minAge As Integer
Dim maxAge As Integer
         ..
minAge = 18
maxAge = 99
rvAge.MinimumValue = minAge.ToString()
rvAge.MaximumValue = maxage.ToString()
```

```
int minAge;
int maxAge;
         ..
   minAge = 18;
maxAge = 99;
   rvAge.MinimumValue = minAge.ToString();
   rvAge.MaximumValue = maxAge.ToString();
```

If control attributes are first set in the *.aspx* file and later set in the code-behind, the values set in the code-behind will be the values used in the rendered page.

The error message to be output when validation fails is placed between the open and close tags:

```
<asp:RangeValidator id="rvAge" Runat="server"
  ControlToValidate="txtAge"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"
   MinimumValue="18"
   MaximumValue="99"
  Type="Integer">
```

```
   <img src="images/arrow_alert.gif" alt="arrow"/>
   Age Must Be Between 18 and 99
 </asp:RangeValidator>
```

The error message that will be output when a validation error occurs can be set dynamically in the code-behind, which is something you might find useful when you have set the maximum and minimum range values on the fly:

VB

```
 rvAge.Text = "<img src='images/arrow_alert.gif' alt="arrow"/> " & _
     "Age Must Be Between " & minAge.ToString() & _
     " and " & maxage.ToString()
```

C#

```
 rvAge.Text = "<img src='images/arrow_alert.gif' alt="arrow"/> " +
     "Age Must Be Between " + minAge.ToString() +
     " and " + maxAge.ToString();
```

If you want a text box to be a required field, add a`RequiredFieldValidator` control to the form as well, which is what we have done in our example with the Age text box:

```
 <asp:RequiredFieldValidator id="rfvAge"
   Runat="server"
   ControlToValidate="txtAge"
   CssClass="AlertText"
   Display="Dynamic"
   EnableClientScript="True">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    Age Is Required
 </asp:RequiredFieldValidator>
 <asp:RangeValidator id="rvAge" Runat="server"
   ControlToValidate="txtAge"
   CssClass="AlertText"
   Display="Dynamic"
   EnableClientScript="True"
    MinimumValue="18"
    MaximumValue="99"
   Type="Integer">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    Age Must Be Between 18 and 99
 </asp:RangeValidator>
```

All other aspects of the *.aspx* and code-behind are the same as for Recipe 3.1. See that recipe's discussion for comments about the `Display`, `EnableClientScript`, and `CausesValidation` attributes in particular. You'll also find explanations of the Save and Cancel buttons and various other aspects of the code.

See Also

Recipe 3.1

Example 3-4. Form with range validation (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
    AutoEventWireup="false"
     CodeFile="CH03RangeValidationVB.aspx.vb"
      Inherits="ASPNetCookbook.VBExamples.CH03RangeValidationVB"
    title="Range Validation" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">

 Range Validation (VB)
  </div>
  <table align="center" class="dataEntry"></tr>
    <td class="LabelText">First Name: </td>
    <td>
      <asp:TextBox id="txtFirstName" Runat="server"
       Columns="30" CssClass="LabelText" />
    </td>
    </tr>
    </tr>
    <td class="LabelText">Last Name: </td>
    <td>
   <asp:TextBox id="txtLastName" Runat="server"
       Columns="30" CssClass="LabelText" />
    </td>
    </tr>
    </tr>
    <td class="LabelText">Age: </td>
    <td>
   <asp:TextBox id="txtAge" Runat="server"
       Columns="30" CssClass="LabelText" />
      <asp:RequiredFieldValidator id="rfvAge"
    Runat="server"
    ControlToValidate="txtAge"
    CssClass="alertText"
    Display="Dynamic"
    EnableClientScript="True">
   <img src="images/arrow_alert.gif" alt="arrow"/>
   Age Is Required
   </asp:RequiredFieldValidator>
   <asp:RangeValidator id="rvAge" Runat="server"
```

```
     ControlToValidate="txtAge"
     CssClass="alertText"
     Display="Dynamic"
     EnableClientScript="True"
     MinimumValue="18"
     MaximumValue="99"
     Type="Integer">
     <img src="images/arrow_alert.gif" alt="arrow"/>
     Age Must Be Between 18 and 99
     </asp:RangeValidator>
     </td>
     </tr>
     </tr>
     <td class="LabelText">Country: </td>
     <td>
    <asp:DropDownList id="ddCountry" Runat="server" >
    <asp:ListItem Selected="True"
      Value="0">----- Select Country -----</asp:ListItem>
     <asp:ListItem Value="1">Canada</asp:ListItem>
     <asp:ListItem Value="2">United States</asp:ListItem>
       </asp:DropDownList>

     </td>
  </tr>
     </tr>
     <td class="LabelText">Email Address: </td>
     <td>
     <asp:TextBox id="txtEmailAddress" Runat="server"
       Columns="30" CssClass="LabelText" />
     </td>
  </tr>
  </tr>
     <td class="LabelText">Password: </td>
     <td>
      <asp:TextBox id="txtPassword1" Runat="server"
      TextMode="Password"
      Columns="30" CssClass="LabelText" />          </td>
     </tr>
     </tr>
     <td class="LabelText">Re-enter Password: </td>
     <td>
      <asp:TextBox id="txtPassword2" Runat="server"
      TextMode="Password"
       Columns="30" CssClass="LabelText" />
     </td>
     </tr>
     </tr>
      <td colspan="2">
    <br/>
     <table align="center" width="50%"></tr>
     <td align="center">
     <input id="btnSave" runat="server" type="button"
     value="Save" causesvalidation="true"
     onserverclick="btnSave_ServerClick"/>
     </td>
     <td align="center">
          <input id="btnCancel" runat="server" type="button"
                 value="Cancel" causesvalidation="false"
     onserverclick="btnCancel_ServerClick" />
         </td>
      </tr>
     </table>
    </td>
    </tr>
    </table>
 </asp:Content>
```

PREV

Recipe 3.4. Requiring That Two Data Input Fields Match

Problem

You need to make sure the data a user enters in two fields on an input form is the same, such as when performing password or email verification.

Solution

Add `RequiredFieldValidator` controls to the *.aspx* file for both `TextBox` controls to prevent a user from skipping one of the fields. Next, add a `CompareValidator` control to one of the `TextBox` controls. Finally, verify that validation was successful from within the event handler of the control that completes the user's entry for the page.

In the *.aspx* file:

1. Add a `RequiredFieldValidator` control for each of the two text boxes in which the user must enter matching data.
2. Add a `CompareValidator` control to the control that must have its input match the other control (usually the second one).
3. Add Save and Cancel (or equivalently named) buttons.
4. Set the Save button's `CausesValidation` attribute to `true` to have validation performed when the button is clicked (set it to `False` for the Cancel button).

In the code-behind class for the page, use the .NET language of your choice to add code to the event handler for the Save button's click event to check the `Page.IsValid` property and verify that all validation was successful. (See Recipe 3.1 for details.)

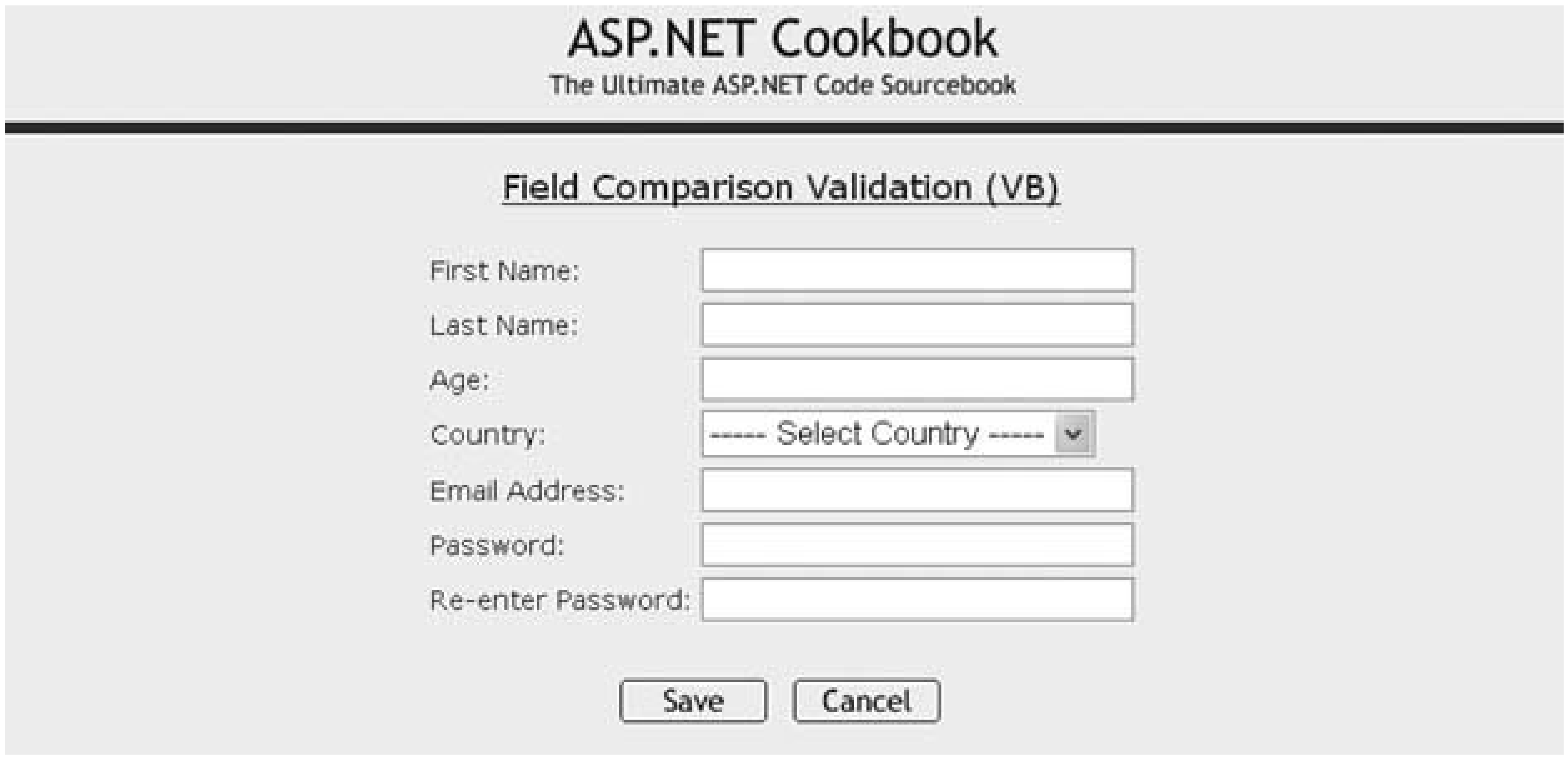
Figure 3-5 shows a typical form with normal output prior to data entry. Figure 3-6 shows the error message that appears on the form when the user enters passwords that do not match. Example 3-5 shows the *.aspx* file for the solution we have implemented to illustrate this recipe. See Examples 3-2 and 3-3 (Recipe 3.1) for the companion code-behind files.

Discussion

The first step in making sure the data a user enters in two fields is the same is to place `RequiredFieldValidator` controls in the form for the two fields. In our application, for example, `RequiredFieldValidator` controls are added for the Password and Re-enter Password text boxes.

Next, a CompareValidator control must be added for the second field whose input must match the first field's input. In our example application, aCompareValidator is added for the Re-enter Password because its input must match the Password's input. The validators are placed to the right of the text boxes to cause the error messages to be displayed beside the text boxes.

Figure 3-5. Form with compare validation outputnormal



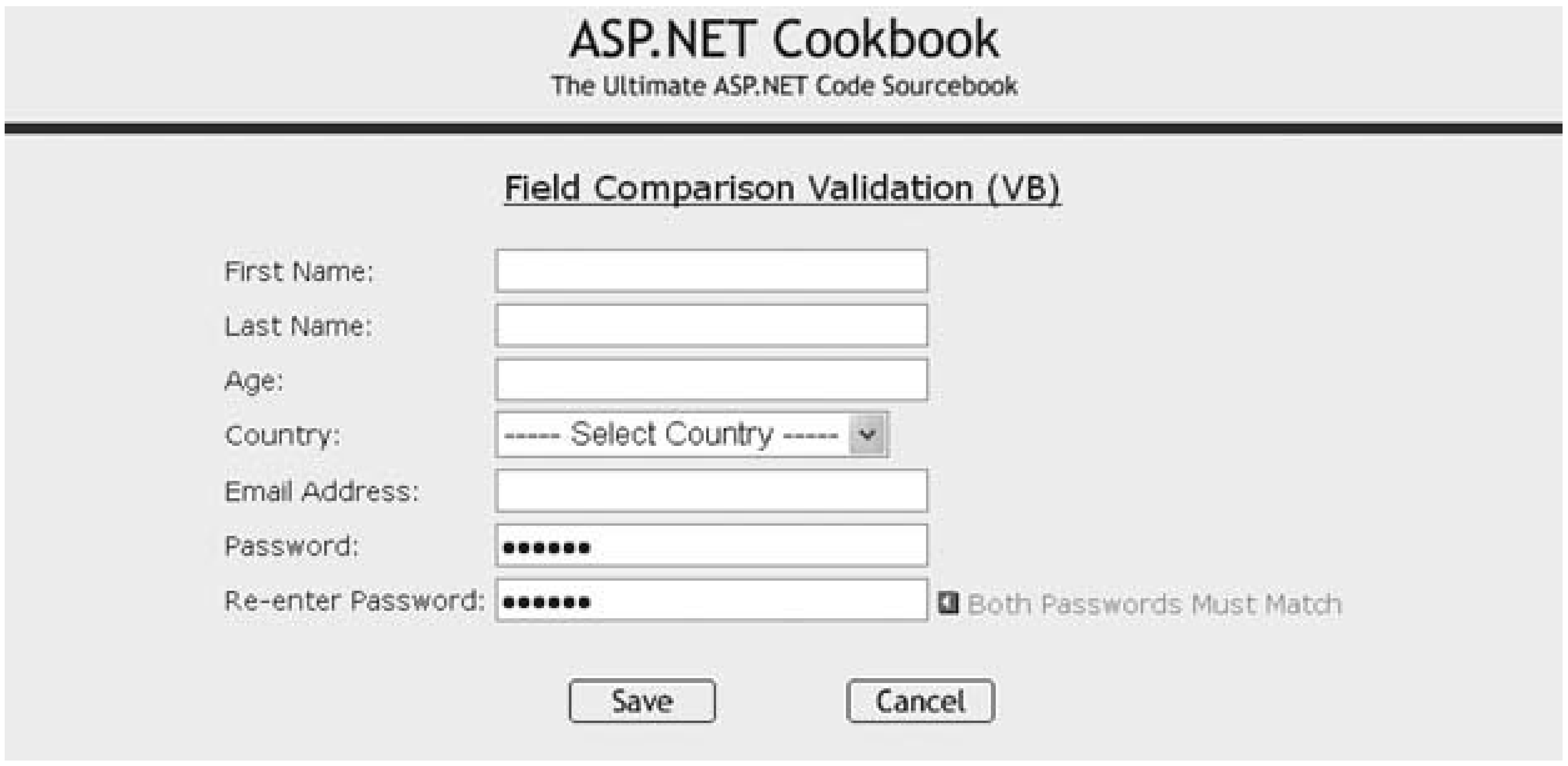
ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

Field Comparison Validation (VB)

First Name:	
Last Name:	
Age:	
Country:	----- Select Country -----
Email Address:	
Password:	
Re-enter Password:	

Figure 3-6. Form with compare validation outputwith error message


ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

Field Comparison Validation (VB)

First Name:	
Last Name:	
Age:	
Country:	----- Select Country -----
Email Address:	
Password:	••••••
Re-enter Password:	•••••• Both Passwords Must Match

> Because the `CompareValidator` allows empty input, a `RequiredFieldValidator` must be used with each of the controls involved in the comparison. Attempting to use a `CompareValidator` without a `RequiredFieldValidator` will result in odd and undesirable behavior.
>
> In our application, for example, if the `RequiredFieldValidator` were omitted for the Re-enter Password text box, the comparison validation would be performed only if data were entered in the Re-enter Password text box. The end result would be failing to flaga validation error when the user enters data in the Password text box but leaves the Re-enter Password text box empty.

See Recipe 3.1 for more information about how to set up therequired field validators.

The `ControlToValidate` attribute of the `CompareValidator` control is set to the ID of the control to validate. On our form, the control to validate is `txtPassword2`, the `TextBox` in which the user re-enters his password:

```
<asp:CompareValidator  ID="cvPassword2"  runat="server"
  ControlToValidate="txtPassword2"
   ControlToCompare="txtPassword1"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True">
   <img src="images/arrow_alert.gif" alt="arrow"/>
   Both Passwords Must Match
</asp:CompareValidator>
```

The ControlToCompare attribute is set to the ID of the control used as the reference, in our case txtPassword1:

```
<asp:CompareValidator  ID="cvPassword2"  runat="server"
   ControlToValidate="txtPassword2"
  ControlToCompare="txtPassword1"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True">
      <img src="images/arrow_alert.gif" alt="arrow"/>
   Both Passwords Must Match
</asp:CompareValidator>
```

In all other respects, the *.aspx* and code-behind files are the same as those in Recipe 3.1. See that recipe's discussion for comments about the Display, EnableClientScript, and CausesValidation attributes in particular. You'll also find explanations of the Save and Cancel buttons and various othe aspects of the code.

See Also

Recipe 3.1

Example 3-5. Form with compare validation (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH03CompareValidationVB.aspx.vb"
   Inherits="ASPNetCookbook.VBExamples.CH03CompareValidationVB"
  title="Compare Validator" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
   <div align="center" class="pageHeading">
     Field Comparison Validation (VB)
   </div>
   <table align="center" class="dataEntry"></tr>
  <td class="LabelText">First Name: </td>
  <td>
    <asp:TextBox id="txtFirstName" Runat="server"
     Columns="30" CssClass="LabelText" />

  </td>
     </tr>
     </tr>
  <td class="LabelText">Last Name: </td>
  <td>
```

```
    <asp:TextBox id="txtLastName" Runat="server"
     Columns="30" CssClass="LabelText" />
      </td>
      </tr>
      </tr>
<td class="LabelText">Age: </td>
<td>
    <asp:TextBox id="txtAge" Runat="server"
     Columns="30" CssClass="LabelText" />
      </td>
      </tr>
      </tr>
<td class="LabelText">Country: </td>
<td>
    <asp:DropDownList id="ddCountry" Runat="server" >
    <asp:ListItem Selected="True"
     Value="0">----- Select Country -----</asp:ListItem>
      <asp:ListItem Value="1">Canada</asp:ListItem>
      <asp:ListItem Value="2">United States</asp:ListItem>
    </asp:DropDownList>
      </td>
      </tr>
      </tr>
<td class="LabelText">Email Address: </td>
<td>
    <asp:TextBox id="txtEmailAddress" Runat="server"
       Columns="30" CssClass="LabelText" />
      </td>
      </tr>
      </tr>
<td class="LabelText">Password: </td>
<td>
     <asp:TextBox id="txtPassword1" Runat="server"
      TextMode="Password"
       Columns="30" CssClass="LabelText" />
    <asp:RequiredFieldValidator  id="rfvPassword1"
    Runat="server"
     ControlToValidate="txtPassword1"
    CssClass="alertText"
    Display="Dynamic"
    EnableClientScript="True">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    Password Is Required
  </asp:RequiredFieldValidator>
      </td>
      </tr>
      </tr>

      <td class="LabelText">Re-enter Password: </td>
<td>
     <asp:TextBox id="txtPassword2" Runat="server"
      TextMode="Password"
      Columns="30" CssClass="LabelText" />
    <asp:RequiredFieldValidator  id="rvPassword2"
    Runat="server"
     ControlToValidate="txtPassword2"
    CssClass="alertText"
    Display="Dynamic"
    EnableClientScript="True">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    Re-Entered Password Is Required
    </asp:RequiredFieldValidator>
    <asp:CompareValidator ID="cvPassword2" runat="server"
    ControlToValidate="txtPassword2"
    ControlToCompare="txtPassword1"
    CssClass="alertText"
    Display="Dynamic"
    EnableClientScript="True">
     <img src="images/arrow_alert.gif" alt="arrow"/>
     Both Passwords Must Match
       </asp:CompareValidator>
     </td>
     </tr>
     </tr>
    <td colspan="2">
   <br/>
   <table align="center" width="50%"></tr>
       <td align="center">
       <input id="btnSave" runat="server" type="button"
       value="Save" causesvalidation="true"
       onserverclick="btnSave_ServerClick"/>
          </td>
          <td align="center">
          <input id="btnCancel" runat="server" type="button"
       value="Cancel" causesvalidation="false"
       onserverclick="btnCancel_ServerClick" />
     </td>
     </tr>
         </table>
      </td>
   </tr>
   </table>
</asp:Content>
```

Recipe 3.5. Requiring Data to Match a Predefined Pattern

Problem

You need to make sure the data a user enters matches a specific pattern, such as an email address.

Solution

Add a `RegularExpressionValidator` control to the *.aspx* file, set the regular expression to perform the pattern matching, and verify that validation was successful from within the event handler of the control that completes the user's entry for the page.

In the *.aspx* file:

1. Add a `RegularExpressionValidator` control for each text box that must have data matching a specific pattern.
2. Set the `ValidationExpression` attribute of the `RegularExpressionValidator` to the regular expression needed to match the required pattern.
3. Add Save and Cancel (or equivalently named) buttons.
4. Set the Save button's `CausesValidation` attribute to `true` to have validation performed when the button is clicked (set it to `False` for the Cancel button).

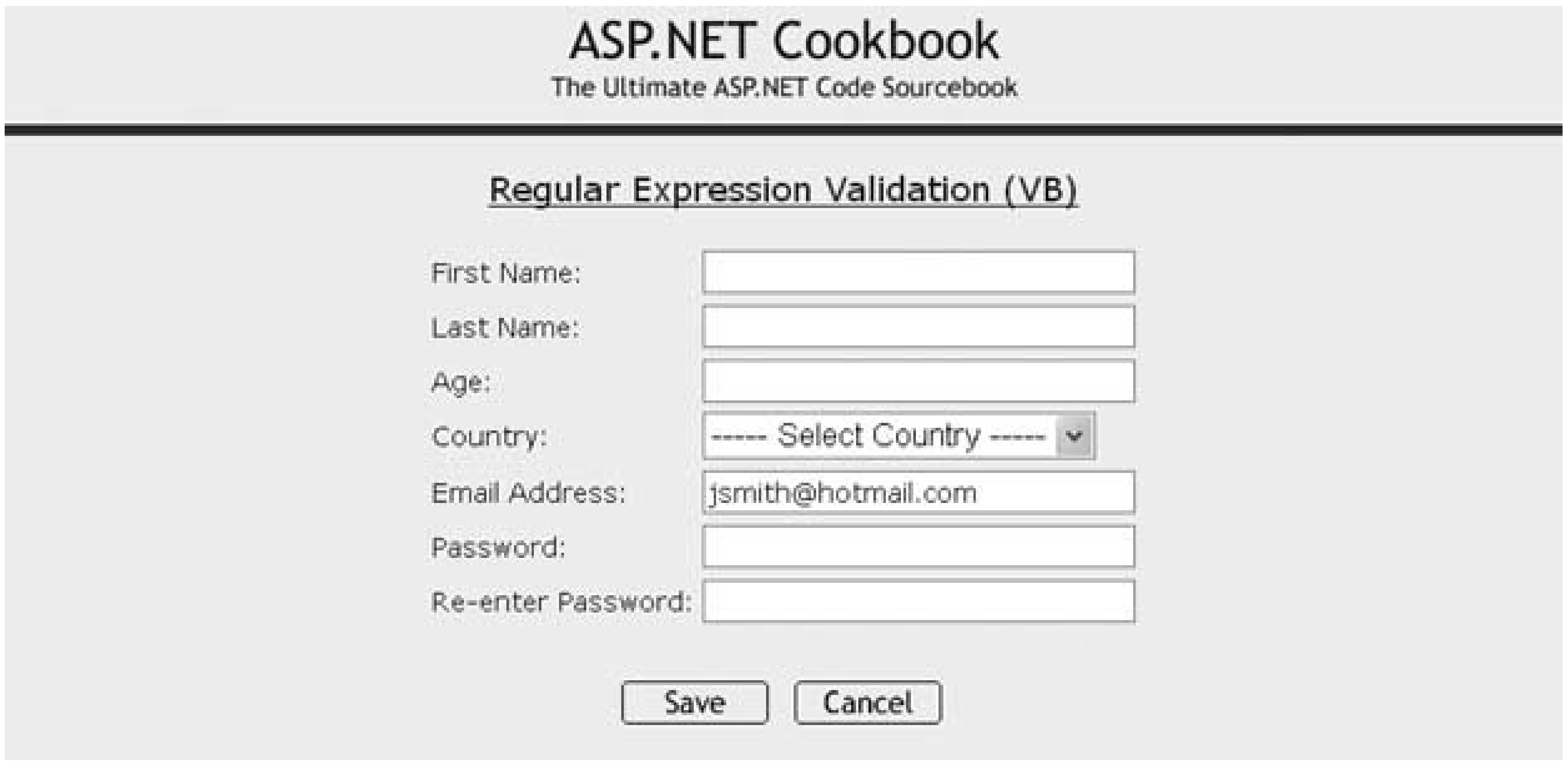
In the code-behind class for the page, use the .NET language of your choice to add code to the event handler for the Save button's click event to check the `Page.IsValid` property and verify that all validation was successful. (See Recipe 3.1 for details.)

Figure 3-7 shows a typical form with normal, error-free output. Figure 3-8 shows the error message that appears on the form when an invalid email address is entered. Example 3-6 shows the *.aspx* file for our application that implements the recipe. (See Examples 3-2 and 3-3 [Recipe 3.1] for the companion code-behind files.)

Discussion

One of the more common uses of pattern validation is for checking the form of an email address entered to ensure it matches the *user@domain* pattern. A `RegularExpressionValidator` control is added for the Email Address text box in the example. The control is placed to the right of the Email Address text box and causes the error message to be displayed beside the text box when an invalid email address is entered.

Figure 3-7. Form with pattern validation outputnormal


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Regular Expression Validation (VB)

First Name:	
Last Name:	
Age:	
Country:	----- Select Country -----
Email Address:	jsmith@hotmail.com
Password:	
Re-enter Password:	

Figure 3-8. Form with pattern validationwith error message

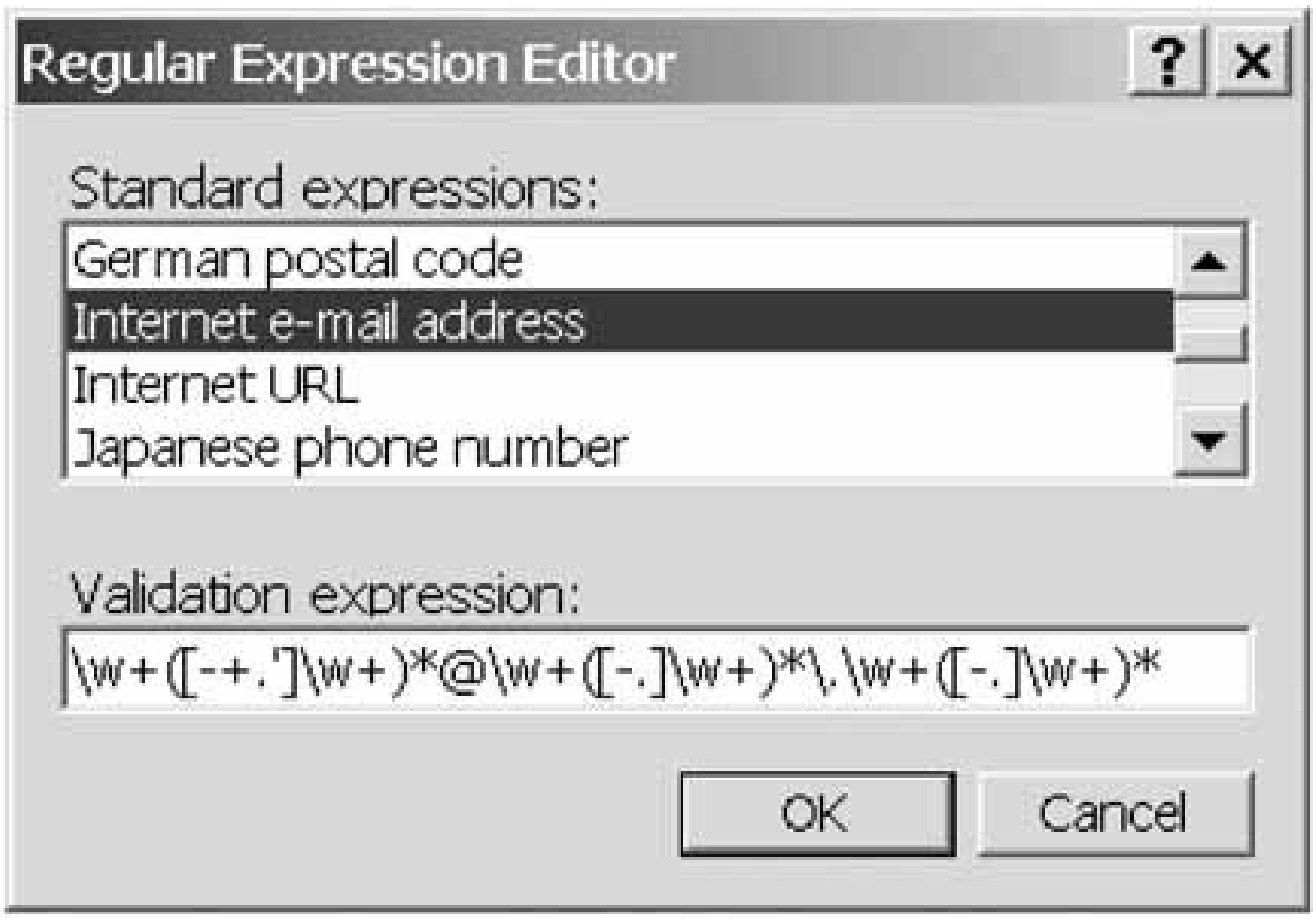
The `ControlToValidate` attribute of the validation control must be set to the ID of the control to validate, in our case, `txtEmailAddress`:

```
<asp:RegularExpressionValidator id="revEmailAddress"
  Runat="server"
  ControlToValidate="txtEmailAddress"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"

  ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
   <img src="images/arrow_alert.gif" alt="arrow"/>
   Invalid Email Address
  </asp:RegularExpressionValidator>
```

The `ValidationExpression` attribute is set to the regular expression that will perform the pattern matching on the data entered into the text box. Any valid regular expression can be used. The expression we use in our example is the standard prebuilt expression for an Internet email address chosen from a pick list in the Regular Expression Editor of Visual Studio 2005, which is available whe setting the `ValidateExpression` attribute (see Figure 3-9). The Help accessible from this same dialog box provides a complete explanation of the syntax and can be used when writing your own custom regular expressions. Many books describe all the nuances of regular expressions, including *Mastering Regular Expressions* (O'Reilly), so we won't go into them here.

```
<asp:RegularExpressionValidator id="revEmailAddress"
  Runat="server"
  ControlToValidate="txtEmailAddress"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"
  ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
   <img src="images/arrow_alert.gif" alt="arrow"/>
   Invalid Email Address
</asp:RegularExpressionValidator>
```

Figure 3-9. Regular Expression Editor



\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*

The error message to be output when validation fails is placed between the open and close tags. It can include HTML, as shown here:

```
<asp:RegularExpressionValidator id="revEmailAddress"
  Runat="server"
  ControlToValidate="txtEmailAddress"
  CssClass="AlertText"

  Display="Dynamic"
  EnableClientScript="True"
  ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
   <img src="images/arrow_alert.gif" alt="arrow"/>
   Invalid Email Address
</asp:RegularExpressionValidator>
```

If the email address in this example was required to process the form, a `RequiredFieldValidator` would need to be added along with the `RegularExpressionValidator`.

In all other respects, the *.aspx* and code-behind files are identical to those in Recipe 3.1. See that recipe's discussion for comments about the `Display`, `EnableClientScript`, and `CausesValidation` attributes in particular. You'll also find explanations of the Save and Cancel buttons and various othe aspects of the code.

See Also

Recipe 3.1; search *Regular Expression Examples* in the MSDN Library; *Mastering Regular Expressions*, by Jeffrey E. F. Friedl (O'Reilly); the Help available from the Regular Expression dialog box when setting the `ValidateExpression` attribute in Visual Studio 2005.

Example 3-6. Form with pattern validation (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
   AutoEventWireup="false"
     CodeFile="CH03RegularExpressionValidationVB.aspx.vb"
      Inherits="ASPNetCookbook.VBExamples.CH03RegularExpressionValidationVB"
   title="Regular Expression Validator" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
   <div align="center" class="pageHeading">
   Regular Expression Validation (VB)
   </div>
   <table align="center" class="dataEntry"></tr>
    <td class="LabelText">First Name: </td>
    <td>
    <asp:TextBox id="txtFirstName" Runat="server"
     Columns="30" CssClass="LabelText" />
   </td>
  </tr>
  </tr>
   <td class="LabelText">Last Name: </td>
   <td>
   <asp:TextBox id="txtLastName" Runat="server"
     Columns="30" CssClass="LabelText" />
   </td>
  </tr>
  </tr>
   <td class="LabelText">Age: </td>
   <td>

         <asp:TextBox id="txtAge" Runat="server"
     Columns="30" CssClass="LabelText" />
  </td>
  </tr>
  </tr>
   <td class="LabelText">Country: </td>
   <td>
   <asp:DropDownList id="ddCountry" Runat="server" >
     <asp:ListItem Selected="True"
     Value="0">----- Select Country -----</asp:ListItem>
      <asp:ListItem Value="1">Canada</asp:ListItem>
      <asp:ListItem Value="2">United States</asp:ListItem>
      </asp:DropDownList>
```

```
 </td>
</tr>
</tr>
 <td class="LabelText">Email Address: </td>
 <td>
  <asp:TextBox id="txtEmailAddress" Runat="server"
    Columns="30" CssClass="LabelText" />
  <asp:requiredfieldvalidator id="rfvEmailAddress"
      runat="server"
      controltovalidate="txtEmailAddress"
      cssclass="alertText"
      display="Dynamic"
      enableclientscript="True">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    Email Address Is Required
  </asp:requiredfieldvalidator>
  <asp:RegularExpressionValidator id="revEmailAddress"
       Runat="server"
      ControlToValidate="txtEmailAddress"
      CssClass="alertText"
      Display="Dynamic"
      EnableClientScript="True"
    ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    Invalid Email Address
  </asp:RegularExpressionValidator>
   </td>
</tr>
</tr>
 <td class="LabelText">Password: </td>
 <td>
   <asp:TextBox id="txtPassword1" Runat="server"
    TextMode="Password"
     Columns="30" CssClass="LabelText" />
   </td>
</tr>
</tr>
 <td class="LabelText">Re-enter Password: </td>
 <td>

  <asp:TextBox id="txtPassword2" Runat="server"
    TextMode="Password"
     Columns="30" CssClass="LabelText" />
 </td>
</tr>
</tr>
    <td colspan="2">
   <br/>
    <table align="center" width="50%"></tr>
    <td align="center">
   <input id="btnSave" runat="server" type="button"
    value="Save" causesvalidation="true"
```

```
     onserverclick="btnSave_ServerClick"/>
  </td>
  <td align="center">
    <input id="btnCancel" runat="server" type="button"
     value="Cancel" causesvalidation="false"
     onserverclick="btnCancel_ServerClick" />
      </td>
     </tr>
     </table>
  </td>
  </tr>
   </table>
</asp:Content>
```

PREV

PREV

Recipe 3.6. Requiring That a Drop-Down List Selection Be Made

Problem

You need to make sure a user selects an entry in a drop-down list.

Solution

Add a `CustomValidator` control to the drop-down list, along with some client-side JavaScript to validate the selection. Next, implement an event handler for the`CustomValidator` control's `ServerValidate` event. Finally, check the `Page.IsValid` property in the event handler for the control that completes the user's entry for the page.

In the *.aspx* file:

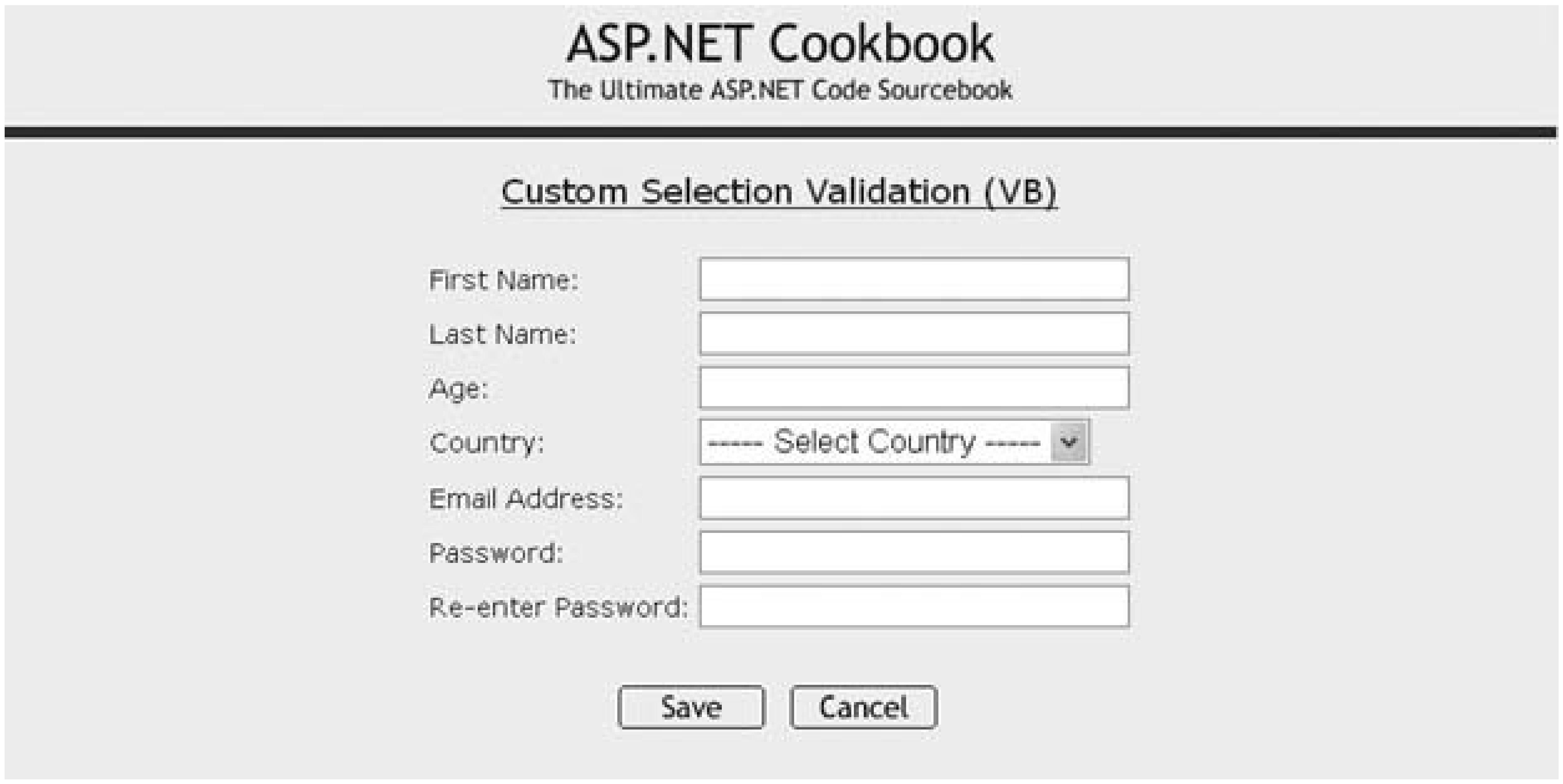
1. Add a `CustomValidator` control for each drop-down list where you must verify that an item has been selected.
2. Add JavaScript to validate the selection on the client side.
3. Add Save and Cancel (or equivalently named) buttons.
4. Set the Save button's `CausesValidation` attribute to `true` to have validation performed when the button is clicked (set it to `False` for the Cancel button).

In the code-behind class for the page, use the .NET language of your choice to:

1. Add an event handler for the `CustomValidator` control's `ServerValidate` event whose purpose is to provide the server-side validation to ensure an item has been selected.
2. Add code to the event handler for the Save button's click event to check the`Page.IsValid` property and verify that all validation was successful (see Recipe 3.1 for details).

Figure 3-10 shows a typical form with normal output prior to data entry. Figure 3-11shows the form with validation errors. Examples 3-7 , 3-8 through 3-9 show the *.aspx* and code-behind files for our application that implements the solution.

Figure 3-10. Form with selection validation outputnormal



ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

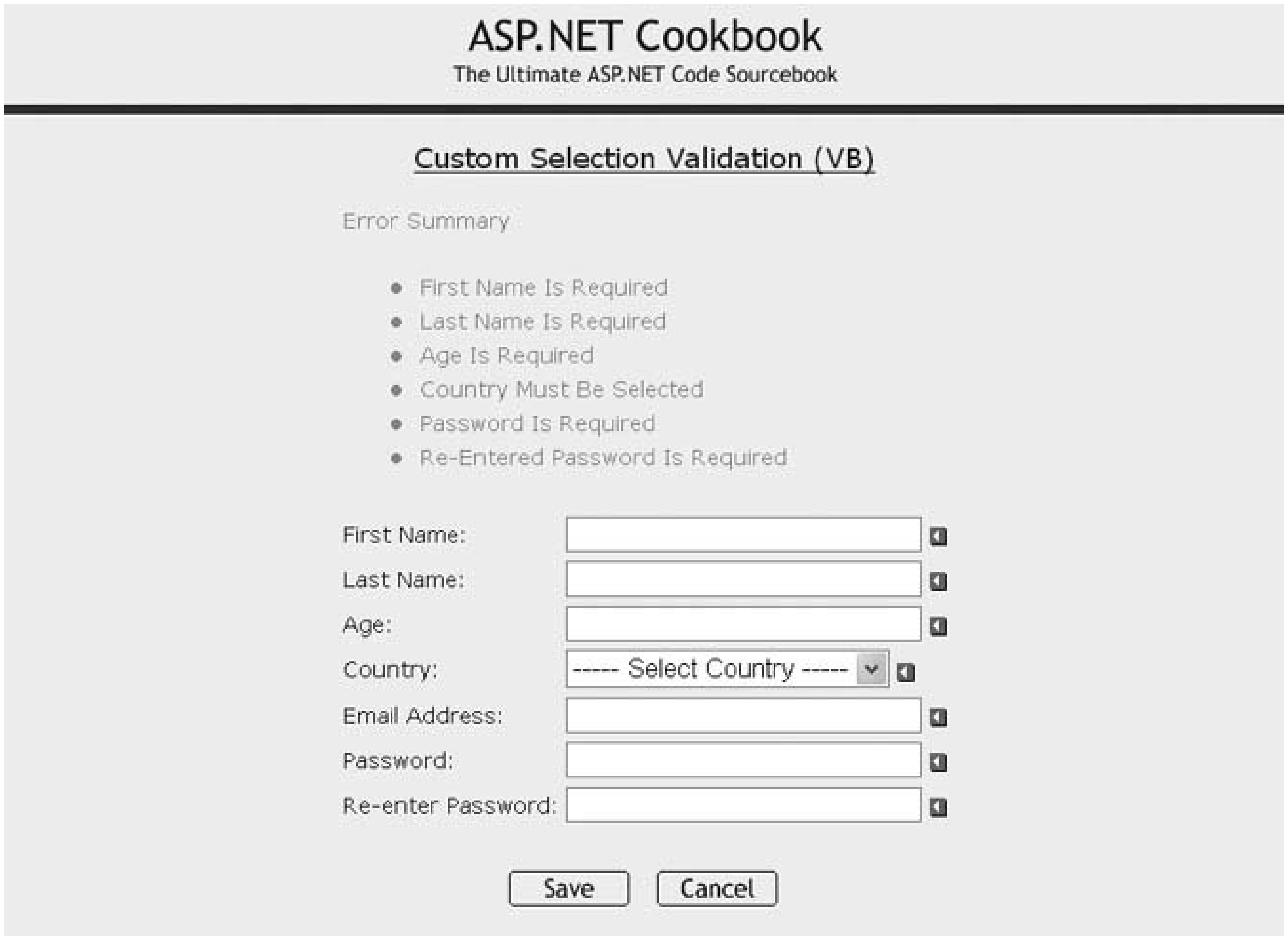
Custom Selection Validation (VB)

First Name:	
Last Name:	
Age:	
Country:	----- Select Country -----
Email Address:	
Password:	
Re-enter Password:	

Discussion

This recipe involves using a `CustomValidator` control to verify that an item has been selected in a drop-down list. But the approach we advocate is out of the ordinary for a couple of reasons. First, by implementing validation via client-side JavaScript, errors can be detected on the client, thus avoiding unnecessary round trips to the server for server-side validation.

Figure 3-11. Form with selection validation outputwith error summary


ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

Custom Selection Validation (VB)

Error Summary

- First Name Is Required
- Last Name Is Required
- Age Is Required
- Country Must Be Selected
- Password Is Required
- Re-Entered Password Is Required

First Name:	
Last Name:	
Age:	
Country:	----- Select Country -----
Email Address:	
Password:	
Re-enter Password:	

Besides using a CustomValidator control, this application uses validation controls from all of the previous recipes combined, specifically RequiredFieldValidator , RangeValidator , CompareValidator , and RegularExpressionValidator . Combining validators of various types is typical when performing validation on a complex form.

Second, instead of displaying an error message at each control, this example shows how to use a ValidationSummary control to provide a list of all errors on the page in one place. This approach is useful for a "busy" form.

To implement the solution, the ControlToValidate attribute must be set to the drop-down list you will be validating. In our case, it is set to ddCountry :

```
<asp:CustomValidator id="valItemSelected" runat="server"
  ControlToValidate="ddCountry"
  ClientValidationFunction="isItemSelected"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"
  ErrorMessage="Country Must Be Selected"
```

```
    OnServerValidate="valItemSelected_ServerValidate">
    <img src="images/arrow_alert.gif" alt="arrow"/>
  </asp:CustomValidator>
```

The ClientValidationFunction attribute must be set to the name of the client JavaScript function that will perform the client-side validation, which is done to ensure an item has been selected from the drop down:

```
  <asp:CustomValidator id="valItemSelected" runat="server"
    ControlToValidate="ddCountry"
    ClientValidationFunction="isItemSelected"
    CssClass="AlertText"
    Display="Dynamic"
    EnableClientScript="True"
    ErrorMessage="Country Must Be Selected"
    OnServerValidate="valItemSelected_ServerValidate">
     <img src="images/arrow_alert.gif" alt="arrow"/>
  </asp:CustomValidator>
```

A client script block must be added to the page containing the function named in the ClientValidationFunction attribute of the CustomValidator , as shown in Example 3-7. The function must have source and argument parameters. When the function is called, the source will be set to a reference to the validator that called the function. In this case, it is the "valItemSelected " validator.

The arguments parameter is a structure containing two elements: Value and IsValid. Value contains the current value of the control being validated. In this case, it is the value of the selected item in th drop-down list. In our example, three entries are added to the drop-down list, with the first entry being the Select Country instruction with a value of 0 . All legitimate selections from the drop-down contain values greater than 0; therefore, if the value of arguments.Value is less than 1, no selection has been made and the value of arguments.IsValid is set to False to indicate a validation failure. If arguments.Value is greater than 0, then a selection has been made, and arguments.IsValid is set to TRue to indicate the validation passed.

The EnableClientScript attribute is set to true or False according to how you want validation to be performed. Setting the attribute to true causes validation to be performed on the client and on the server when the page is submitted. Setting the attribute to False causes validation to be performed only on the server when the form is submitted. In our example, we are providing client-side JavaScript, so it must be set to true :

```
  <asp:CustomValidator id="valItemSelected" runat="server"
    ControlToValidate="ddCountry"
    ClientValidationFunction="isItemSelected"
    CssClass="AlertText"
    Display="Dynamic"
    EnableClientScript="True"
    ErrorMessage="Country Must Be Selected"
```

```
   OnServerValidate="valItemSelected_ServerValidate">
    <img src="images/arrow_alert.gif" alt="arrow"/>
</asp:CustomValidator>
```

Rather than outputtin gan error message at each control, as was done in all the previous examples ir this chapter, we've added a `ValidationSummary` control to the form in this example to provide in one place a summary of all the errors on the form. When a validation summary is being used, an error message is no longer required between the start and end tags of the validator element. Instead, the `ErrorMessage` attribute of the validator is set to the error message to display. To provide visual feedback of which control has an error, an arrow image is inserted between the start and end tags of the validator element in our example:

```
<asp:CustomValidator id="valItemSelected" runat="server"
  ControlToValidate="ddCountry"
   ClientValidationFunction="isItemSelected"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"
  ErrorMessage="Country Must Be Selected"
   OnServerValidate="valItemSelected_ServerValidate">
  <img src="images/arrow_alert.gif" alt="arrow"/>
</asp:CustomValidator>
```

The `DisplayMode` attribute of the `asp:ValidationSummary` control defines how the summary is displayed. Valid values are `BulletList, List` , and `SingleParagraph` . `BulletList` will generate a bulleted errors list, which is what we've chosen for our example. `List` will generate the same list as the `BulletList` setting but without the bullets. `SingleParagraph` generates a single HTML paragraph containing all of the error information. The `HeaderText` attribute is set to the title placed at the top of the errors list.

```
<asp:ValidationSummary id="vsErrors" Runat="server"
  CssClass="AlertText"
  DisplayMode="BulletList"
  EnableClientScript="True"
  HeaderText="Error Summary" />
```

The sample application's code-behind includes an event handler for the `CustomValidator` control's `ServerValidate` event, as shown in Examples 3-8 (VB) and 3-9 (C#). This event handler provides the server-side validation to ensure a country has been selected using the same technique implemented in the client script.

The code-behind includes an event handler for the Save button's click event. This event handler checks to ensure the page is valid (all validation passed) and then performs the processing of the form data.

See Also

Recipes 3.1, 3.2, 3.3, and 3.4

Example 3-7. Form with selection validation (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
     AutoEventWireup="false"
   CodeFile="CH03CustomSelectionValidationVB.aspx.vb"

  Inherits="ASPNetCookbook.VBExamples.CH03CustomSelectionValidationVB"
     title="Custom Selection Validator" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <script language="javascript" type="text/javascript">
 <!--
    function isItemSelected(source, arguments)
    {
   if (arguments.Value < 1)
   {
   arguments.IsValid = false;
   }
   else
   {
   arguments.IsValid = true;
   }
    }
   //-->
   </script>
   <div align="center" class="pageHeading">
     Custom Selection Validation (VB)
   </div>
   <table align="center" class="dataEntry">
  </tr>
    <td colspan="2" align="left">
     <asp:ValidationSummary id="vsErrors" Runat="server"
       CssClass="alertText"
       DisplayMode="BulletList"
       EnableClientScript="True"
       HeaderText="Error Summary" />
   </td>
  </tr>
  </tr>
   <td class="LabelText">First Name: </td>
   <td>
    <asp:TextBox id="txtFirstName" Runat="server"
     Columns="30" CssClass="LabelText" />
    <asp:RequiredFieldValidator id="rfvFirstName"
    Runat="server"
    ControlToValidate="txtFirstName"
    CssClass="alertText"
    Display="Dynamic"
    EnableClientScript="True"
    ErrorMessage="First Name Is Required">
    <img src="images/arrow_alert.gif" alt="arrow"/>
    </asp:RequiredFieldValidator>
   </td>
  </tr>
  <tr>
   <td class="LabelText">Last Name: </td>
   <td>
    <asp:TextBox id="txtLastName" Runat="server"
          Columns="30" CssClass="LabelText" />

    <asp:RequiredFieldValidator id="rfvLastName"
     Runat="server"
     ControlToValidate="txtLastName"
     CssClass="alertText"
     Display="Dynamic"
     EnableClientScript="True"
     ErrorMessage="Last Name Is Required">
     <img src="images/arrow_alert.gif" alt="arrow"/>
    </asp:RequiredFieldValidator>
   </td>
  </tr>
  </tr>
   <td class="LabelText">Age: </td>
   <td>
    <asp:TextBox id="txtAge" Runat="server"
      Columns="30" CssClass="LabelText" />
     <asp:RequiredFieldValidator id="Requiredfieldvalidator1"
     Runat="server"
     ControlToValidate="txtAge"
     CssClass="alertText"
     Display="Dynamic"
     EnableClientScript="True"
     ErrorMessage="Age Is Required">
     <img src="images/arrow_alert.gif" alt="arrow"/>
    </asp:RequiredFieldValidator>
       <asp:RangeValidator id="rvAge" Runat="server"
     ControlToValidate="txtAge"
     CssClass="alertText"
     Display="Dynamic"
     EnableClientScript="True"
      MinimumValue="18"
      MaximumValue="99"
     Type="Integer"
      ErrorMessage="Age Must Be Between 18 and 99">
     <img src="images/arrow_alert.gif" alt="arrow"/>
       </asp:RangeValidator>
   </td>
  </tr>
```

```
</tr>
 <td class="LabelText">Country: </td>
 <td>
  <asp:DropDownList id="ddCountry" Runat="server" >
   <asp:ListItem Selected="True"
   Value="0">----- Select Country -----</asp:ListItem>
    <asp:ListItem Value="1">Canada</asp:ListItem>
    <asp:ListItem Value="2">United States</asp:ListItem>
   </asp:DropDownList>
   <asp:CustomValidator id="valItemSelected" runat="server"
       ControlToValidate="ddCountry"
       ClientValidationFunction="isItemSelected"
       CssClass="alertText"
       Display="Dynamic"

       EnableClientScript="True"
       ErrorMessage="Country Must Be Selected"
       OnServerValidate="valItemSelected_ServerValidate">
   <img src="images/arrow_alert.gif" alt="arrow"/>
   </asp:CustomValidator>
 </td>
</tr>
</tr>
   <td class="LabelText">Email Address: </td>
   <td>
  <asp:TextBox id="txtEmailAddress" Runat="server"
       Columns="30" CssClass="LabelText" />
     <asp:requiredfieldvalidator id="rfvEmailAddress"
  runat="server"
  controltovalidate="txtEmailAddress"
  cssclass="alertText"
  display="Dynamic"
  enableclientscript="True">
    <img src="images/arrow_alert.gif" alt="arrow"/>
  </asp:requiredfieldvalidator>
  <asp:RegularExpressionValidator id="revEmailAddress"
   Runat="server"
   ControlToValidate="txtEmailAddress"
   CssClass="alertText"
   Display="Dynamic"
   EnableClientScript="True"
  ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
         ErrorMessage="Invalid Email Address">
    <img src="images/arrow_alert.gif" alt="arrow"/>
     </asp:RegularExpressionValidator>
   </td>
</tr>
</tr>
   <td class="LabelText">Password: </td>
   <td>
      <asp:TextBox id="txtPassword1" Runat="server"
    TextMode="Password"
```

```
      Columns="30" CssClass="LabelText" />
    <asp:RequiredFieldValidator id="rfvPassword1"
   Runat="server"
    ControlToValidate="txtPassword1"
   CssClass="alertText"
   Display="Dynamic"
   EnableClientScript="True"
   ErrorMessage="Password Is Required">
    <img src="images/arrow_alert.gif" alt="arrow"/>
   </asp:RequiredFieldValidator>
     </td>
 </tr>
 </tr>
    <td class="LabelText">Re-enter Password: </td>
    <td>

    <asp:TextBox id="txtPassword2" Runat="server"
     TextMode="Password"
      Columns="30" CssClass="LabelText" />
    <asp:RequiredFieldValidator id="rvPassword2"
   Runat="server"
    ControlToValidate="txtPassword2"
   CssClass="alertText"
   Display="Dynamic"
   EnableClientScript="True"
   ErrorMessage="Re-Entered Password Is Required">
    <img src="images/arrow_alert.gif" alt="arrow"/>
   </asp:RequiredFieldValidator>
    <asp:CompareValidator ID="cvPassword2" runat="server"
    ControlToValidate="txtPassword2"
    ControlToCompare="txtPassword1"
   CssClass="alertText"
   Display="Dynamic"
   EnableClientScript="True"
   ErrorMessage="Both Passwords Must Match">
    <img src="images/arrow_alert.gif" alt="arrow"/>
     </asp:CompareValidator>
  </td>
 </tr>
 </tr>
    <td colspan="2">
    <br/>
     <table align="center" width="50%">
    </tr>
   <td align="center">
   <input id="btnSave" runat="server" type="button"
     value="Save" causesvalidation="true"
     onserverclick="btnSave_ServerClick"/>
   </td>
   <td align="center">
       <input id="btnCancel" runat="server" type="button"
     value="Cancel" causesvalidation="false"
```

```
      onserverclick="btnCancel_ServerClick" />
      </td>
       </tr>
      </table>
     </td>
    </tr>
   </table>
</asp:Content>
```

Example 3-8. Form with selection validation code-behind (.vb)

```
Option Explicit On
Option Strict On
Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
  '''  CH03CustomSelectionValidationVB.aspx

  ''' </summary>
  Partial  Class  CH03CustomSelectionValidationVB
 Inherits System.Web.UI.Page
 '''**************************************************************************
    ''' <summary>
 ''' This routine provides the event handler for the valItemSelected
 ''' server validate event. It is responsible for validating that a
 ''' country has been selected.
 ''' </summary>
 '''
 ''' <param name="source">Set to the sender of the event</param>
 ''' <param name="args">Set to the event arguments</param>
 Protected Sub valItemSelected_ServerValidate(ByVal source As Object, _
    ByVal args As System.Web.UI.WebControls.ServerValidateEventArgs)
    If (ddCountry.SelectedIndex < 1) Then
    args.IsValid = False
    Else
    args.IsValid = True
    End If
 End Sub 'valItemSelected_ServerValidate
 '''**************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the save button click
 ''' event. It is responsible for processing the form data.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnSave_ServerClick(ByVal sender As Object, _
         ByVal e As System.EventArgs)
   If (Page.IsValid) Then
    'process form data and save as required for application
   End If
 End Sub 'btnSave_ServerClick
 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the cancel button click
 ''' event. It is responsible for processing the form data.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnCancel_ServerClick(ByVal sender As Object, _
           ByVal e As System.EventArgs)
  'perform any cancel operations required for application
 End Sub 'btnCancel_ServerClick
   End Class 'CH03CustomSelectionValidationVB
End Namespace
```

Example 3-9. Form with selection validation code-behind (.cs)

```
using System;
namespace ASPNetCookbook.CSExamples
{
   /// <summary>
   /// This class provides the code behind for
    /// CH03CustomSelectionValidationCS.aspx
   /// </summary>
    public partial class CH03CustomSelectionValidationCS
   : System.Web.UI.Page
   {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the valItemSelected server
  /// validate event. It is responsible for validating that a country has
  /// been selected
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void valItemSelected_ServerValidate(Object source,
         System.Web.UI.WebControls.ServerValidateEventArgs args)
  {
    if (ddCountry.SelectedIndex < 1)
  {
   args.IsValid = false;
  }
  else
  {
   args.IsValid = true;
  }
   } // valItemSelected_ServerValidate
   ///***********************************************************************
   /// <summary>
   /// This routine provides the event handler for the save button click
   /// event. It is responsible for processing the form data.
   /// </summary>
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void btnSave_ServerClick(Object sender,
            System.EventArgs e)
   {
  if (Page.IsValid)
  {
     // process form data and save as required for application
  }
   } //btnSave_ServerClick
   ///***********************************************************************
   /// <summary>
   /// This routine provides the event handler for the cancel button click
   /// event. It is responsible for processing the form data.
   /// </summary>

   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void btnCancel_ServerClick(Object sender,
         System.EventArgs e)
   {
  // perform any cancel operations required for application
   } // btnCancel_ServerClick
  }  //CH03CustomSelectionValidationCS
}
```

Recipe 3.7. Requiring Data to Match a Database Entry

Problem

You need to make sure the data a user enters matches an entry in a database.

Solution

Add a `CustomValidator` to the *.aspx* file. Then add an event handler to the codebehind for the `CustomValidator` control's `ServerValidate` event, its purpose beingto validate the user entries against the database.

In the *.aspx* file:

1. Add a `CustomValidator` control that validates the entries against the database during server-side validation.
2. Add a Login (or equivalently named) button.

In the code-behind class for the page, use the .NET language of your choice to:

1. Add an event handler for the `CustomValidator` control's `ServerValidate` event, its purpose beingto provide the server-side validation of the user's entries against the database.
2. Add code to the event handler for the Login button's click event to check the`Page.IsValid` property and verify that all validation was successful (see Recipe 3.1 for details).

Figure 3-12 shows a typical form with normal output prior to data entry. Figure 3-13 shows the form with a validation error message. Examples 3-10, 3-11 through 3-12 show the *.aspx* and code-behind files for our application that implements the solution.

Discussion

One of the most common examples of this recipe's handiness is when implementing a classic login page. The approach we favor in this scenario uses a`CustomValidator` to perform the user authentication and a `ValidationSummary` to display error information.

Figure 3-12. Form with database validation outputnormal

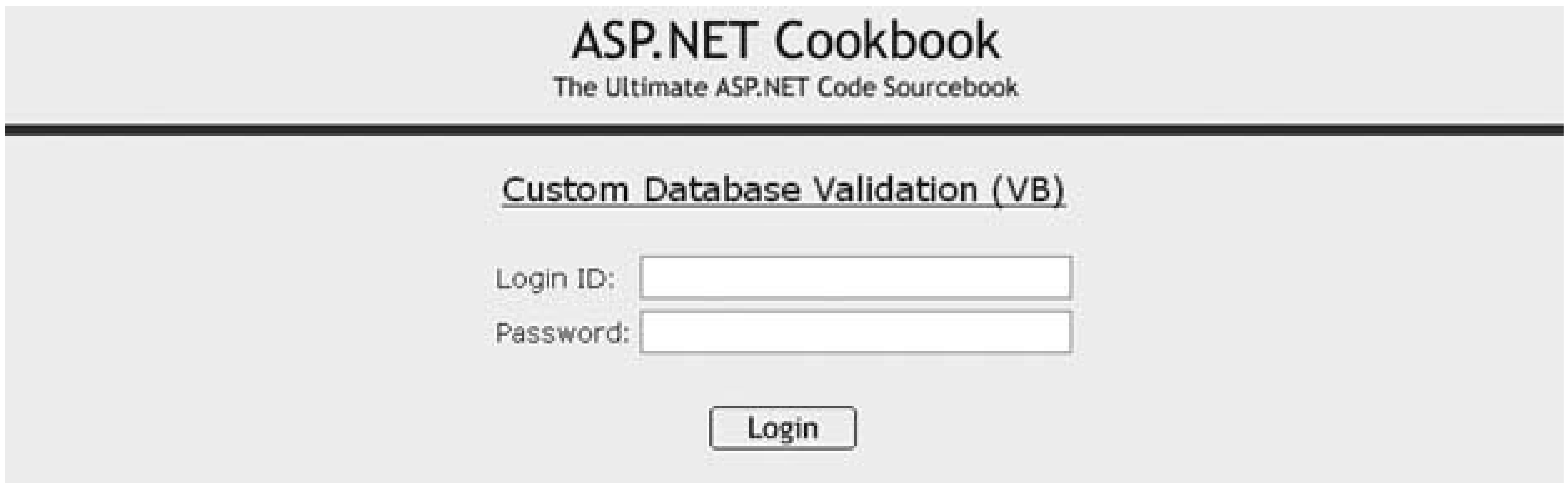
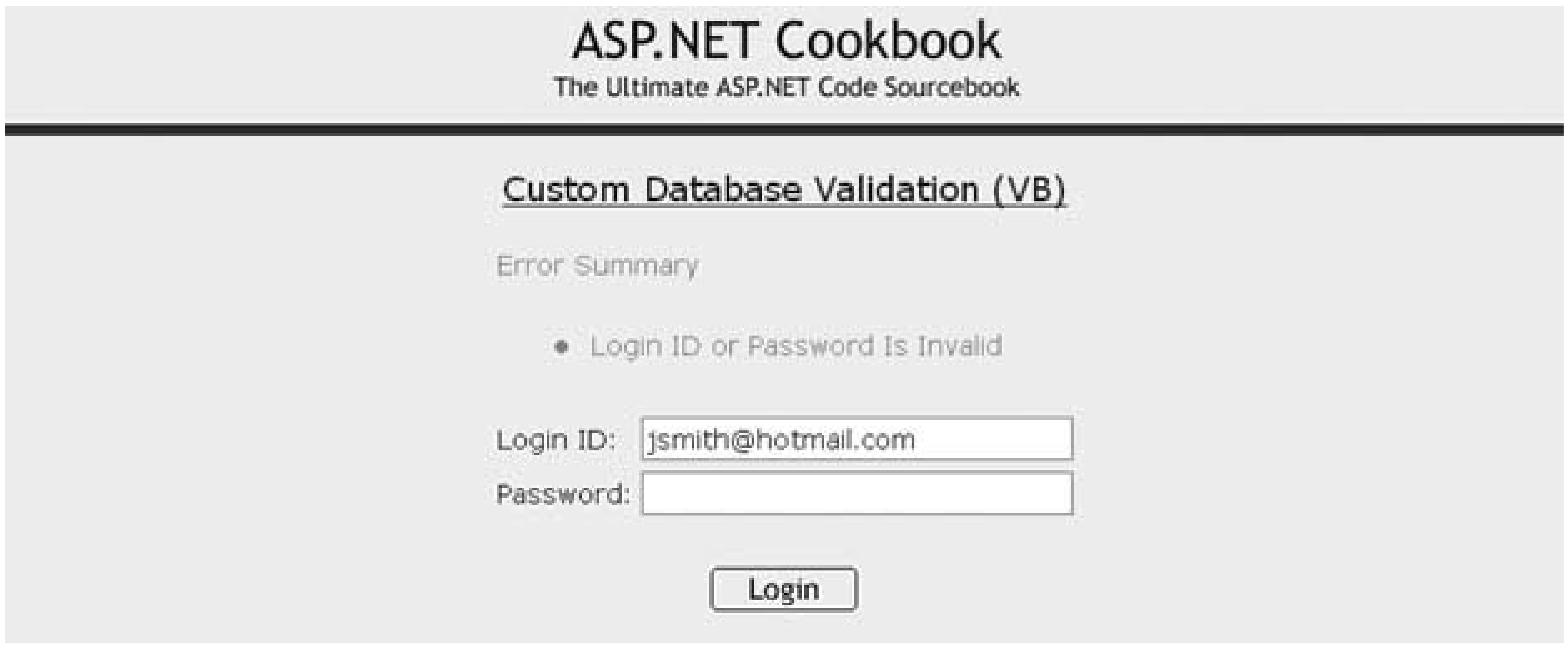

ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Custom Database Validation (VB)
Login ID:
Password:
Login

Figure 3-13. Form with database validation outputwith error message


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Custom Database Validation (VB)
Error Summary
Login ID or Password Is Invalid
Login ID:
jsmith@hotmail.com
Password:
Login

In our example, RequiredFieldValidator controls are used for the login ID and password fields. (RequiredFieldValidator controls are described in Recipe 3.1.) The user must supply both to gain access to her account.

Unlike the other recipes in this chapter, our approach for this recipe has theCustomValidator control's EnableClientScript attribute set to False to disable client side validation because the database validation can be done only on the server side:

```
<asp:CustomValidator id="cvAuthentication" Runat="server"
  Display="None"
  EnableClientScript="False"
  ErrorMessage="Login ID or Password Is Invalid"
  OnServerValidate="cvAuthentication_ServerValidate" />
```

The ValidationSummary is set up to display all validation errors. This includes errors from the RequiredFieldValidator controls and the CustomValidator used for user authentication.

The ServerValidate event for the CustomValidator (cvAuthentication_ServerValidate) is used to perform the user authentication by checkingif a user exists in the database with the entered login ID and password, as shown in Examples 3-11 (VB) and 3-12 (C#).

If the user is found in the database, the args.IsValid property is set true to indicate the validation was successful. Otherwise, it is set False to indicate the validation failed.

The event handler for the Login button's click event (btnLogin_Click) then checks to see if the page is valid before proceeding with actions required to log the user into the system.

As you may have noticed, the approach used in this recipe is an amalgam of all the approaches used in the chapter's other recipes. Having used this approach to control essentially all the aspects of validation, you can adapt it to perform almost any validation your application requires.

See Also

Recipes 3.1 and 3.5

Example 3-10. Form with database validation (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH03CustomDatabaseValidationVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH03CustomDatabaseValidationVB"
  title="Custom Database Validation" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
   <div align="center" class="pageHeading">
  Custom Selection Validation (VB)
   </div>
   <table align="center" class="dataEntry">
   </tr>
    <td colspan="2" align="left">
      <asp:ValidationSummary id="vsErrors" Runat="server"
      CssClass="alertText"
      DisplayMode="BulletList"
      EnableClientScript="True"
      HeaderText="Error Summary" />
   <asp:CustomValidator id="cvAuthentication" Runat="server"
      Display="None"
      EnableClientScript="False"
      ErrorMessage="Login ID or Password Is Invalid"
      OnServerValidate="cvAuthentication_ServerValidate" />
   </td>
   </tr>
   </tr>
   <td class="LabelText">Login ID: </td>
   <td>
   <asp:TextBox id="txtLoginID" Runat="server"
       Columns="30" CssClass="LabelText" />
   <asp:RequiredFieldValidator id="rfvLoginID"
       Runat="server"
       ControlToValidate="txtLoginID"
       CssClass="alertText"

       Display="Dynamic"
       EnableClientScript="True"
       ErrorMessage="Login ID Is Required">
    <img src="images/arrow_alert.gif" alt="arrow"/>
   </asp:RequiredFieldValidator>
   </td>
   </tr>
   </tr>
      <td class="LabelText">Password: </td>
   <td>
   <asp:TextBox id="txtPassword" Runat="server"
    TextMode="Password"
     Columns="30" CssClass="LabelText" />
   <asp:RequiredFieldValidator id="rfvPassword"
       Runat="server"
       ControlToValidate="txtPassword"
       CssClass="alertText"
       Display="Dynamic"
       EnableClientScript="True"
       ErrorMessage="Password Is Required">
     <img src="images/arrow_alert.gif" alt="arrow"/>
    </asp:RequiredFieldValidator>
     </td>
  </tr>
  </tr>
      <td colspan="2" align="center">
     <br/>
     <input id="btnLogin" runat="server" type="button"
    value="Login" causesvalidation="true"
    onserverclick="btnLogin_Click"/>
    </td>
   </tr>
   </table>
</asp:Content>
```

Example 3-11. Form with database validation code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports System.Configuration.ConfigurationManager
Imports System.Data
Imports System.Data.OleDb
Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
   '''  CH03CustomDatabaseValidationVB.aspx
  ''' </summary>
  Partial Class CH03CustomDatabaseValidationVB
 Inherits System.Web.UI.Page
 '''*************************************************************************
 ''' <summary>

 ''' This routine provides the event handler for the authentication server
 ''' validate event. It is responsible for checking the login ID and
 ''' password in the database to authenticate the user.
 ''' </summary>
 '''
 ''' <param name="source">Set to the sender of the event</param>
 ''' <param name="args">Set to the event arguments</param>
 Protected Sub cvAuthentication_ServerValidate(ByVal source As Object, _
       ByVal args As ServerValidateEventArgs)
    Dim dbConn As OleDbConnection = Nothing
    Dim dCmd As OleDbCommand = Nothing
    Dim strConnection As String
    Dim strSQL As String
    Try
   'initially assume credentials are invalid
   args.IsValid = False
   'get the connection string from web.config and open a connection
   'to the database
   strConnection = _
   ConnectionStrings("dbConnectionString").ConnectionString
   dbConn = New OleDb.OleDbConnection(strConnection)
   dbConn.Open( )
   'build the query string and check to see if a user with the entered
   'credentials exists in the database
   strSQL = "SELECT AppUserID FROM AppUser " & _
     "WHERE LoginID=? AND " & _
     "Password=?"
   dCmd = New OleDbCommand(strSQL, dbConn)
   dCmd.Parameters.Add(New OleDbParameter("LoginID", _
         txtLoginID.Text))
   dCmd.Parameters.Add(New OleDbParameter("Password", _
         txtPassword.Text))
   'check to see if the user was found
   If (Not IsNothing(dCmd.ExecuteScalar( ))) Then
    args.IsValid = True
      End If
    Finally
   'cleanup
   If (Not IsNothing(dbConn)) Then
     dbConn.Close( )
   End If
     End Try
    End Sub 'cvAuthentication_ServerValidate
    '''*************************************************************************
    ''' <summary>
    ''' This routine provides the event handler for the login button click

    ''' event. It is responsible for processing the form data.
    ''' </summary>
    '''
    ''' <param name="sender">Set to the sender of the event</param>
    ''' <param name="e">Set to the event arguments</param>
    Protected Sub btnLogin_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
   If (Page.IsValid) Then
      'user has been authenticated so proceed with allowing access
      'to the site
   End If
  End Sub 'btnLogin_Click
  End  Class  'CH03CustomDatabaseValidationVB
End Namespace
```

Example 3-12. Form with database validation code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
   ///  CH03CustomDatabaseValidationCS.aspx
  /// </summary>
   public partial class CH03CustomDatabaseValidationCS : System.Web.UI.Page
  {
 ///*************************************************************************
    /// <summary>
 /// This routine provides the event handler for the authentication server
 /// validate event. It is responsible checking the login ID and password
 /// in the database to authenticate the user.
 /// </summary>
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
```

```
protected void cvAuthentication_ServerValidate(Object source,
   System.Web.UI.WebControls.ServerValidateEventArgs args)
{
   OleDbConnection dbConn = null;
   OleDbCommand dCmd = null;
   String strConnection = null;
   String strSQL = null;
   try
   {
    // initially assume credentials are invalid
  args.IsValid = false;

  // get the connection string from web.config and open a connection
     // to the database
  strConnection = ConfigurationManager.
  ConnectionStrings["dbConnectionString"].ConnectionString;
  dbConn = new OleDbConnection(strConnection);
  dbConn.Open( );
  // build the query string and check to see if a user with the
  // entered credentials exists in the database
  strSQL = "SELECT AppUserID FROM AppUser " +
   "WHERE LoginID=? AND " +
   "Password=?";
  dCmd = new OleDbCommand(strSQL, dbConn);
  dCmd.Parameters.Add(new OleDbParameter("LoginID",
          txtLoginID.Text));
  dCmd.Parameters.Add(new OleDbParameter("Password",
          txtPassword.Text));
  // check to see if the user was found
  if (dCmd.ExecuteScalar( ) != null)
  {
  args.IsValid = true;
  }
   } // try
   finally
   {
  // cleanup
  if (dbConn != null)
  {
    dbConn.Close( );
  }
   } // finally
 } // cvAuthentication_ServerValidate
 ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the login button click
 /// event. It is responsible for processing the form data.
 /// </summary>
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnLogin_Click(Object sender,
       System.EventArgs e)
  {
    if (Page.IsValid)
    {
   // user has been authenticated so proceed with allowing access
   // to the site
    }
  } //btnLogin_Click
   } //  CH03CustomDatabaseValidationCS
}
```

PREV

PREV

Recipe 3.8. Using Validation Groups to Support Login and New User Registration Within a Single Form

Problem

You have two logical sections on a form with controls that require different validation as a function of the button the user clicks, such as a form that supportslogin and new user registration.

Solution

For each group of controls and its associated form-submit button, set the`ValidationGroup` attribute to a unique group name and set the `CausesValidatio` attribute of the form-submit buttons to `TRue` .

In the *.aspx* file:

1. Set the `ValidationGroup` attribute of a form-submit button to a unique group name.
2. Set the `ValidationGroup` attribute of each of the controls to be validated when the button is clicked to the same group name as the form-submit button.
3. Set the form-submit button's `CausesValidation` attribute to True to have validation performed when the button is clicked.
4. Repeat steps 13 for each form-submit button and its associated controls.

In the code-behind class for the page, use the .NET language of your choice to add code to the event handler for each button's click event to check the`Page.IsValid` property and verify that all validation was successful. (See Recipe 3.1 for details.)

Figure 3-14 shows a typical form with normal, error-free output. Figure 3-15shows the error message that appears on the form when a validation error occurs in the login section of the form. Figure 3-16 shows the error message that appears on the form when a validation error occurs in the form's register section. Examples 3-13 , 3-14 through 3-15 show the *.aspx* and code-behind files for our application that implements the solution.

Discussion

In ASP.NET 1.x, handling multiple logical sections on a form with more than one submit button was difficult, especially when you needed different validation to be performed as a function of which button the user clicked. You generally had to resort to custom code to control the validation. Depending on the number of buttons and controls, this could result in some complex code.

Figure 3-14. Form with group validation outputnormal

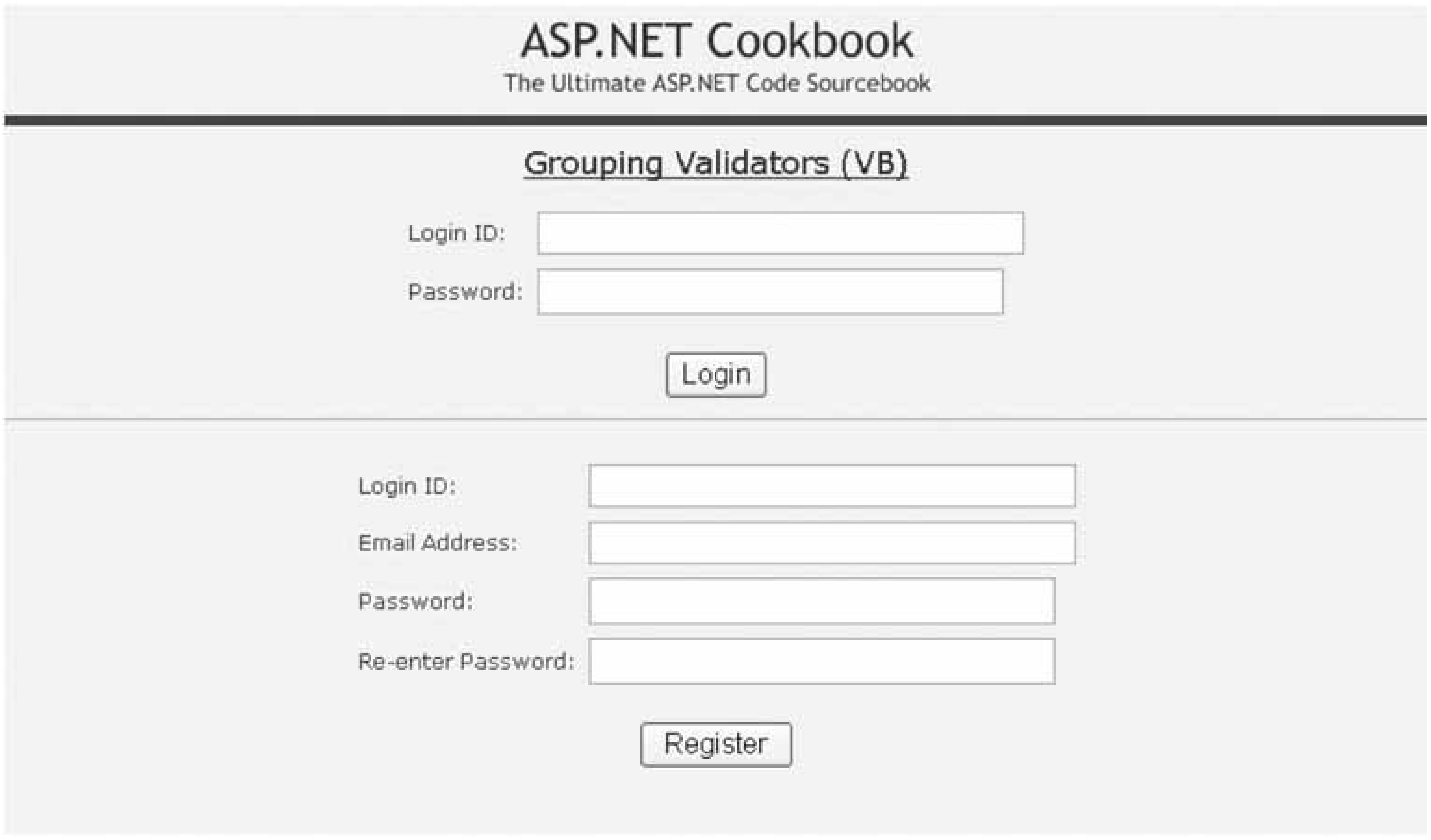
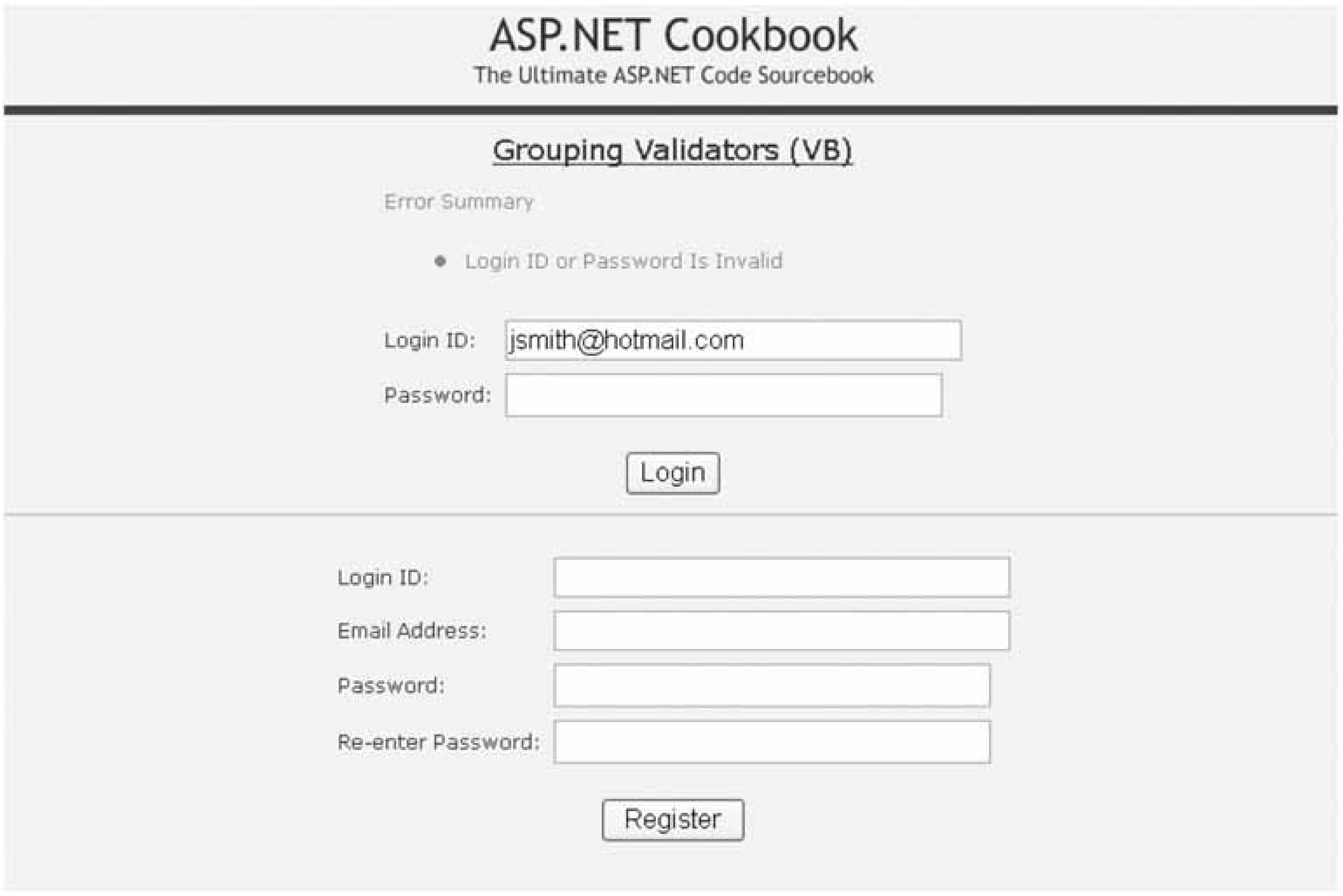


Figure 3-15. Form with group validation outputfirst group failed validatio


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Grouping Validators (VB)

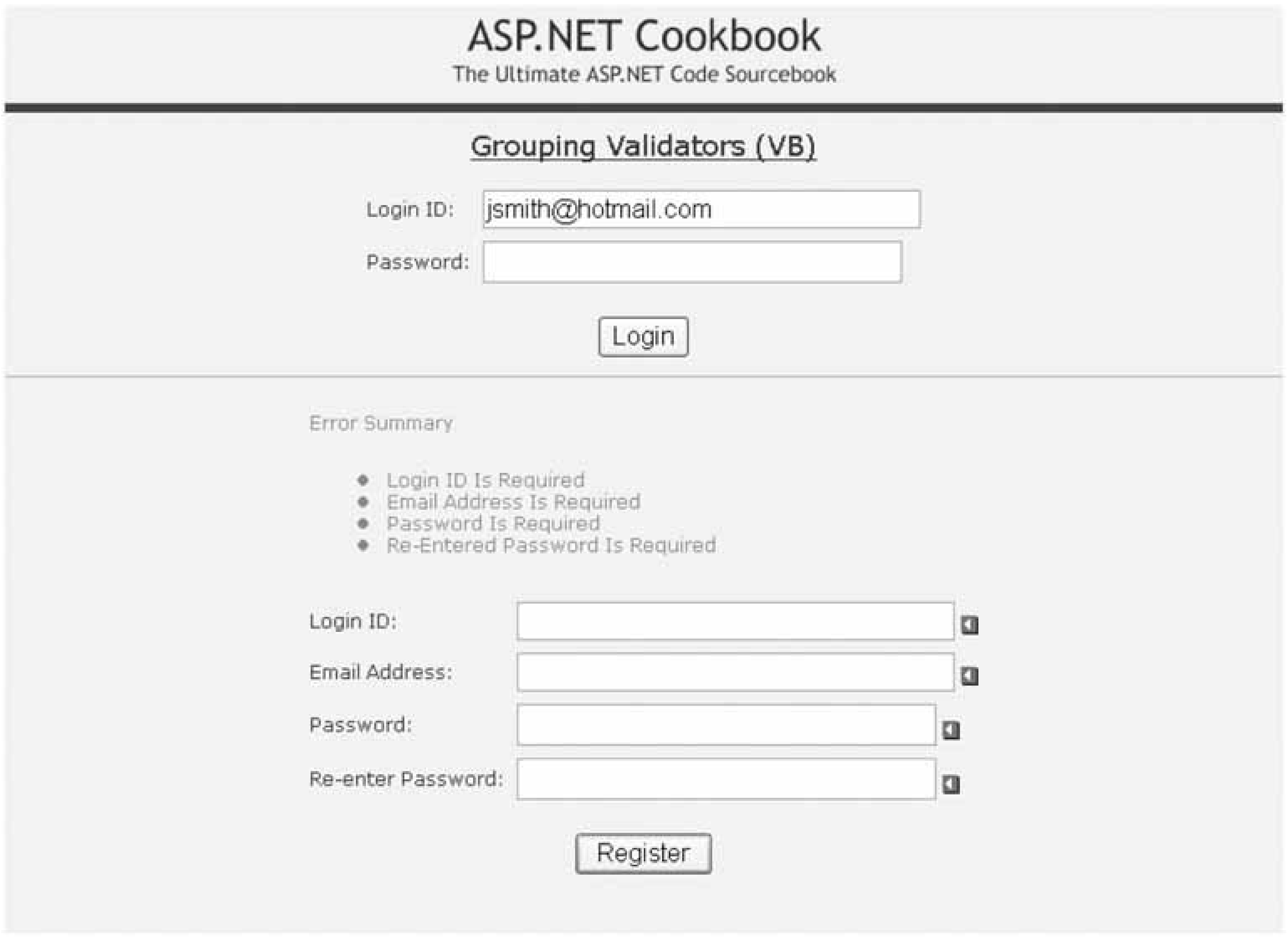
Error Summary

- Login ID or Password Is Invalid

Login ID: jsmith@hotmail.com
Password:

Login ID:
Email Address:
Password:
Re-enter Password:

Figure 3-16. Form with group validation outputsecond group failed validation


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Grouping Validators (VB)

Login ID: jsmith@hotmail.com
Password:

Login

Error Summary

- Login ID Is Required
- Email Address Is Required
- Password Is Required
- Re-Entered Password Is Required

Login ID:
Email Address:
Password:
Re-enter Password:

Register

In ASP.NET 2.0, all validators and form-submit buttons have an optional `ValidationGroup` attribute. Setting The `ValidationGroup` attribute of a submit button, and the controls to be validated when the button is clicked to the same group name, provides the ability to define what controls are validated with each button. This reduces the code required to support multisection forms.

Any controls that do not have the `ValidationGroup` attribute set to a value are grouped by default to provide backward compatibility with ASP.NET 1.x.

Our example application has two sections on the form, one with login controls (login ID, password, and a login button) and the other with registration controls (login ID, email address, password, re-enter password, and a register button).

The `ValidationGroup` attribute for each of the controls in the login section are set to `LoginGroup` . When the Login button is clicked, only the controls with the `ValidationGroup` set to `LoginGroup` are validated.

```
<asp:CustomValidator id="cvAuthentication" Runat="server"
  Display="None"
```

```
    EnableClientScript="False"
    ErrorMessage="Login ID or Password Is Invalid"

    OnServerValidate="cvAuthentication_ServerValidate"
    ValidationGroup="LoginGroup" />
    …
  <asp:RequiredFieldValidator id="rfvLoginID"
    Runat="server"
    ControlToValidate="txtLoginID"
    CssClass="AlertText"
    Display="Dynamic"
    EnableClientScript="True"
    ErrorMessage="Login ID Is Required"
    ValidationGroup="LoginGroup" >
     <img src="images/arrow_alert.gif" alt="arrow"/>
  </asp:RequiredFieldValidator>
    …
  <asp:RequiredFieldValidator id="rfvPassword"
    Runat="server"
    ControlToValidate="txtPassword"
    CssClass="AlertText"
    Display="Dynamic"
    EnableClientScript="True"
    ErrorMessage="Password Is Required"
    ValidationGroup="LoginGroup" >
     <img src="images/arrow_alert.gif" alt="arrow"/>
  </asp:RequiredFieldValidator>
    …
  <input id="btnLogin" runat="server" type="button"
   value="Login" causesvalidation="true"
   onserverclick="btnLogin_Click"
   validationgroup="LoginGroup" >
```

The `ValidationGroup` attribute for each of the controls in the registration section are set to `RegisterGroup` . When the Register button is clicked, only the controls with the`ValidationGroup` set to `RegisterGroup` are validated.

```
  <asp:RequiredFieldValidator id="rfvNewLoginID"
    Runat="server"
    ControlToValidate="txtNewLoginID"
    CssClass="AlertText"
    Display="Dynamic"
    EnableClientScript="True"
    ErrorMessage="Login ID Is Required"
    ValidationGroup="RegisterGroup" >
     <img src="images/arrow_alert.gif" alt="arrow"/>
  </asp:RequiredFieldValidator>
   …
```

```
<asp:requiredfieldvalidator id="rfvEmailAddress"
  runat="server"
  controltovalidate="txtEmailAddress"
  cssclass="AlertText"
  display="Dynamic"
  enableclientscript="True"
  ErrorMessage="Email Address Is Required"
  ValidationGroup="RegisterGroup"  >
   <img src="images/arrow_alert.gif" alt="arrow"/>
</asp:requiredfieldvalidator>
<asp:RegularExpressionValidator id="revEmailAddress"
  Runat="server"
  ControlToValidate="txtEmailAddress"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"
   ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
  ErrorMessage="Invalid Email Address"
  ValidationGroup="RegisterGroup" >
  <img src="images/arrow_alert.gif" alt="arrow"/>
</asp:RegularExpressionValidator>
 …
<asp:RequiredFieldValidator  id="rfvPassword1"
  Runat="server"
   ControlToValidate="txtPassword1"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"
  ErrorMessage="Password Is Required"
  ValidationGroup="RegisterGroup" >
  <img src="images/arrow_alert.gif" alt="arrow"/>
</asp:RequiredFieldValidator>
 …
<asp:RequiredFieldValidator  id="rvPassword2"
  Runat="server"
   ControlToValidate="txtPassword2"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"
  ErrorMessage="Re-Entered Password Is Required"
  ValidationGroup="RegisterGroup" >
   <img src="images/arrow_alert.gif" alt="arrow"/>
</asp:RequiredFieldValidator>
<asp:CompareValidator  ID="cvPassword2"  runat="server"
   ControlToValidate="txtPassword2"
   ControlToCompare="txtPassword1"
  CssClass="AlertText"
  Display="Dynamic"
  EnableClientScript="True"
  ErrorMessage="Both Passwords Must Match"

  ValidationGroup="RegisterGroup" >
```

```
   <img src="images/arrow_alert.gif" alt="arrow"/>
</asp:CompareValidator>
 …
<input id="btnRegister" runat="server" type="button"
  value="Register" causesvalidation="true"
  onserverclick="btnRegister_ServerClick"
  validationgroup="RegisterGroup" >
```

In addition to grouping buttons and user input controls, the`ValidationSummary` control supports the `ValidationGroup` attribute. This provides the ability to use a`ValidationSummary` for each section on the form and to display a summary of the errors in the appropriate section.

```
<asp:ValidationSummary id="vsLoginErrors" Runat="server"
  CssClass="AlertText"
  DisplayMode="BulletList"
  EnableClientScript="True"
  HeaderText="Error Summary"
  ValidationGroup="LoginGroup" />
```

When using validation groups, no custom code is required in client-side JavaScript or in the code-behind to support the validation.

See Also

Recipe 3.8

Example 3-13. Form with multiple validation sections (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
     AutoEventWireup="false"
   CodeFile="CH03GroupingValidatorsVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH03GroupingValidatorsVB"
  title="Grouping Validators" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
   <div align="center" class="pageHeading">
  Grouping Validators (VB)
   </div>
   <table align="center" class="dataEntry">
  </tr>
   <td colspan="2" align="left">
     <asp:ValidationSummary id="vsLoginErrors" Runat="server"
      CssClass="alertText"
      DisplayMode="BulletList"
      EnableClientScript="True"
      HeaderText="Error Summary"
      ValidationGroup="LoginGroup" />
     <asp:CustomValidator id="cvAuthentication" Runat="server"
     Display="None"
     EnableClientScript="False"
     ErrorMessage="Login ID or Password Is Invalid"
     OnServerValidate="cvAuthentication_ServerValidate"
     ValidationGroup="LoginGroup" />
     </td>
  </tr>
  </tr>
  <td class="LabelText">Login ID: </td>
  <td>
      <asp:TextBox id="txtLoginID" Runat="server"
       Columns="30" CssClass="LabelText" />
      <asp:RequiredFieldValidator id="rfvLoginID"
      Runat="server"
      ControlToValidate="txtLoginID"
      CssClass="alertText"
      Display="Dynamic"
      EnableClientScript="True"
      ErrorMessage="Login ID Is Required"
      ValidationGroup="LoginGroup" >
       <img src="images/arrow_alert.gif" alt="arrow"/>
      </asp:RequiredFieldValidator>
   </td>
   </tr>
   </tr>
   <td class="LabelText">Password: </td>
   <td>
      <asp:TextBox id="txtPassword" Runat="server"
        TextMode="Password"
     Columns="30" CssClass="LabelText" />
      <asp:RequiredFieldValidator id="rfvPassword"
      Runat="server"
         ControlToValidate="txtPassword"
      CssClass="alertText"
      Display="Dynamic"
      EnableClientScript="True"
      ErrorMessage="Password Is Required"
      ValidationGroup="LoginGroup" >
      <img src="images/arrow_alert.gif" alt="arrow"/>
      </asp:RequiredFieldValidator>
   </td>
   </tr>
   </tr>
     <td colspan="2" align="center">
   <br/>
   <input id="btnLogin" runat="server" type="button"
       value="Login" causesvalidation="true"
       onserverclick="btnLogin_Click"
       validationgroup="LoginGroup" />
    </td>
   </tr>
      </table>
    <hr />
  <table align="center" class="dataEntry">
   </tr>
     <td colspan="2" align="left">
     <asp:ValidationSummary id="vsRegistrationErrors" Runat="server"
      CssClass="alertText"
      DisplayMode="BulletList"
      EnableClientScript="True"
      HeaderText="Error Summary"
      ValidationGroup="RegisterGroup" />
    </td>
   </tr>
   </tr>
    <td class="LabelText">Login ID: </td>
    <td>
     <asp:TextBox id="txtNewLoginID" Runat="server"
       Columns="30" CssClass="LabelText" />
     <asp:RequiredFieldValidator id="rfvNewLoginID"
     Runat="server"
     ControlToValidate="txtNewLoginID"
     CssClass="alertText"
     Display="Dynamic"
     EnableClientScript="True"
     ErrorMessage="Login ID Is Required"
     ValidationGroup="RegisterGroup" >
     <img src="images/arrow_alert.gif" alt="arrow"/>
     </asp:RequiredFieldValidator>
    </td>
   </tr>
   </tr>
    <td class="LabelText">Email Address: </td>
    <td>
     <asp:TextBox id="txtEmailAddress" Runat="server"
       Columns="30" CssClass="LabelText" />
     <asp:requiredfieldvalidator id="rfvEmailAddress"
     runat="server"
     controltovalidate="txtEmailAddress"
     cssclass="alertText"
     display="Dynamic"
     enableclientscript="True"
     ErrorMessage="Email Address Is Required"
     ValidationGroup="RegisterGroup" >
     <img src="images/arrow_alert.gif" alt="arrow"/>
     </asp:requiredfieldvalidator>
     <asp:RegularExpressionValidator id="revEmailAddress"
     Runat="server"
     ControlToValidate="txtEmailAddress"
     CssClass="alertText"
     Display="Dynamic"
     EnableClientScript="True"
   ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
     ErrorMessage="Invalid Email Address"
     ValidationGroup="RegisterGroup" >
     <img src="images/arrow_alert.gif" alt="arrow"/>
     </asp:RegularExpressionValidator>
    </td>
   </tr>
   </tr>
    <td class="LabelText">Password: </td>
    <td>
      <asp:TextBox id="txtPassword1" Runat="server"
     TextMode="Password"
      Columns="30" CssClass="LabelText" />
     <asp:RequiredFieldValidator id="rfvPassword1"
     Runat="server"
     ControlToValidate="txtPassword1"
     CssClass="alertText"
     Display="Dynamic"
     EnableClientScript="True"
     ErrorMessage="Password Is Required"
     ValidationGroup="RegisterGroup" >
     <img src="images/arrow_alert.gif" alt="arrow"/>
     </asp:RequiredFieldValidator>
    </td>
   </tr>
   </tr>
    <td class="LabelText">Re-enter Password: </td>
    <td>
      <asp:TextBox id="txtPassword2" Runat="server"
      TextMode="Password"
       Columns="30" CssClass="LabelText" />
     <asp:RequiredFieldValidator id="rvPassword2"
     Runat="server"
     ControlToValidate="txtPassword2"
     CssClass="alertText"
     Display="Dynamic"
     EnableClientScript="True"
     ErrorMessage="Re-Entered Password Is Required"
     ValidationGroup="RegisterGroup" >
     <img src="images/arrow_alert.gif" alt="arrow"/>
     </asp:RequiredFieldValidator>
     <asp:CompareValidator ID="cvPassword2" runat="server"
     ControlToValidate="txtPassword2"
     ControlToCompare="txtPassword1"
     CssClass="alertText"
     Display="Dynamic"
     EnableClientScript="True"
     ErrorMessage="Both Passwords Must Match"
     ValidationGroup="RegisterGroup" >
     <img src="images/arrow_alert.gif" alt="arrow"/>
     </asp:CompareValidator>
```

```
   </td>
  </tr>
  </tr>
   <td colspan="2" align="center">
    <br/>
    <input id="btnRegister" runat="server" type="button"
    value="Register" causesvalidation="true"
    onserverclick="btnRegister_ServerClick"
    validationgroup="RegisterGroup" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 3-14. Form with multiple validation sections (.vb)

```
Option Explicit On
Option Strict On
Imports System.Configuration.ConfigurationManager
Imports System.Data
Imports System.Data.OleDb
Namespace ASPNetCookbook.VBExamples
   ''' <summary>
   ''' This class provides the code behind for
    ''' CH03GroupingValidatorsVB.aspx
   ''' </summary>
    Partial Class CH03GroupingValidatorsVB
   Inherits System.Web.UI.Page
    '''*************************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the authentication server
   ''' validate event. It is responsible for checking the login ID and
   ''' password in the database to authenticate the user.
   ''' </summary>
   '''
      ''' <param name="source">Set to the sender of the event</param>
   ''' <param name="args">Set to the event arguments</param>
   Protected Sub cvAuthentication_ServerValidate(ByVal source As Object, _
        ByVal args As ServerValidateEventArgs)
     Dim dbConn As OleDbConnection = Nothing
   Dim dCmd As OleDbCommand = Nothing
   Dim strConnection As String
   Dim strSQL As String
   Try
   'initially assume credentials are invalid
    args.IsValid = False
    'get the connection string from web.config and open a connection
    'to the database
    strConnection = _
     ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDb.OleDbConnection(strConnection)
    dbConn.Open( )
    'build the query string and check to see if a user with the entered
    'credentials exists in the database

    strSQL = "SELECT AppUserID FROM AppUser " & _
    "WHERE LoginID=? AND " & _
    "Password=?"
       dCmd = New OleDbCommand(strSQL, dbConn)
    dCmd.Parameters.Add(New OleDbParameter("LoginID", _
           txtLoginID.Text))
    dCmd.Parameters.Add(New OleDbParameter("Password", _
           txtPassword.Text))
    'check to see if the user was found
    If (Not IsNothing(dCmd.ExecuteScalar( ))) Then
    args.IsValid = True
    End If
   Finally
    'cleanup
    If (Not IsNothing(dbConn)) Then
    dbConn.Close( )
      End If
   End Try
  End Sub 'cvAuthentication_ServerValidate
  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the login button click
  ''' event. It is responsible for processing the form data for login.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnLogin_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)
   If (Page.IsValid) Then
    'user has been authenticated so proceed with allowing access
    'to the site
   End If
  End Sub 'btnLogin_Click
  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the login button click
  ''' event. It is responsible for processing the form data for
  ''' registration.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnRegister_ServerClick(ByVal sender As Object, _
            ByVal e As System.EventArgs)
    If (Page.IsValid) Then
    'all entered data is valid so proceed with registration
    End If

  End Sub 'btnRegister_ServerClick
 End Class 'CH03GroupingValidatorsVB
End Namespace
```

Example 3-15. Form with multiple validation sections (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
namespace ASPNetCookbook.CSExamples
{
   /// <summary>
   /// This class provides the code behind for CH03GroupingValidatorsCS.aspx
   /// </summary>
   public partial class CH03GroupingValidatorsCS
      : System.Web.UI.Page
   {
  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the authentication server
  /// validate event. It is responsible checking the login ID and password
  /// in the database to authenticate the user.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void cvAuthentication_ServerValidate(Object source,
    System.Web.UI.WebControls.ServerValidateEventArgs args)
  {
  OleDbConnection dbConn = null;
  OleDbCommand dCmd = null;
  String strConnection = null;
  String strSQL = null;
  try
  {
    // initially assume credentials are invalid
    args.IsValid = false;
    // get the connection string from web.config and open a connection
    // to the database
    strConnection = ConfigurationManager.
      ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(strConnection);
    dbConn.Open( );
    // build the query string and check to see if a user with the
    // entered credentials exists in the database
    strSQL = "SELECT AppUserID FROM AppUser " +
     "WHERE LoginID=? AND " +
     "Password=?";
    dCmd = new OleDbCommand(strSQL, dbConn);

    dCmd.Parameters.Add(new OleDbParameter("LoginID",
            txtLoginID.Text));
    dCmd.Parameters.Add(new OleDbParameter("Password",
            txtPassword.Text));
    // check to see if the user was found
    if (dCmd.ExecuteScalar( ) != null)
    {
     args.IsValid = true;
    }
   } // try
    finally
    {
   // cleanup
   if (dbConn != null)
   {
    dbConn.Close( );
   }
    } // finally
  } // cvAuthentication_ServerValidate
  ///*******************************************************************
  /// <summary>
  /// This routine provides the event handler for the login button click
  /// event. It is responsible for providing access to the site for the
  /// user if authenticated.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnLogin_Click(Object sender,
          System.EventArgs e)
  {
  if (Page.IsValid)
     {
     // user has been authenticated so proceed with allowing access
     // to the site
   }
  } //btnLogin_Click
  ///*******************************************************************
  /// >summary>
  /// This routine provides the event handler for the register button click
  /// event. It is responsible for processing the form data for
  /// registration.
  /// </summary>
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnRegister_ServerClick(Object sender,
           System.EventArgs e)
  {
    if (Page.IsValid)
  {

   // all entered data is valid so proceed with registration
  }
   } //btnRegister_ServerClick
    } // CH03GroupingValidatorsCS
}
```

PREV

PREV

Recipe 3.9. Performing Validation Programmatically to Execute Your Own Application-Specific Logic

Problem

You have two logical sections on a form with controls that require different validation as a function of the button the user clicks. You want to have programmatic control over the validation performed because your application needs to carry out its own application-specific logic, such as when you want to check a new user's registration against a database.

Solution

For each group of controls and their associated form-submit button, set the`ValidationGroup` attribute to a unique group name. This will cause only the controls with a group name matching the group name of the button to be validated when the button is clicked.

In the *.aspx* file:

1. Set the `ValidationGroup` attribute of a form-submit button to a unique group name.
2. Set the `ValidationGroup` attribute of each of the controls to be validated when the button is clicked to the same group name as the form-submit button.
3. Set the form-submit button's `CausesValidation` attribute to `False` to disable both the client-side and server-side validation performed by ASP.NET when the button is clicked.
4. Repeat steps 13 for each form-submit button and its associated controls.

In the code-behind class for the page, use the .NET language of your choice to:

1. Add an event handler for each form-submit button.
2. Add code to the event handler to get the collection of validators associated with the button.
3. Iterate through the collection performing the required validation.

Figure 3-14 shows a typical form with normal, error-free output. Figure 3-15 shows the error message that appears on the form when a validation error occurs in the login section of the form. Figure 3-16 shows the error message that appears on the form when a validation error occurs in the register section of the form. Examples 3-16, 3-17 through 3-18 show the *.aspx* and code-behind files for our application that implements the solution.

Discussion

Periodically, you must control the validation process that ASP.NET normally provides to perform the validation in a nonstandard way. ASP.NET provides the ability to get a collection containing a specific group of or all validators on a form. A collection of all validators on a form can be obtained as shown below:

VB

```
 Dim allValidators As ValidatorCollection
 …
 allValidators = Page.Validators
```

C#

```
 ValidatorCollection allValidators;

 allValidators = Page.Validators;
```

A collection of validators for a specific group on a form can be obtained as shown below:

```
validators = Page.GetValidators([name of desired group])
```

```
validators = Page.GetValidators([name of desired group]);
```

If `Nothing` (`null` in C#) is passed to the `GetValidators` method, all validators missing the `ValidationGroup` attribute set will be returned.

In our application, we are modifying the code described in Recipe 3.7 to disable the validation provided by ASP.NET by setting the `CausesValidation` attribute of both form-submit buttons to False.

```
 <input id="btnLogin" runat="server" type="button"
   value="Login" CausesValidation="false"
   onserverclick="btnLogin_Click"
   validationgroup="LoginGroup" >
 …
 <input id="btnRegister" runat="server" type="button"
   value="Register" CausesValidation="false"
   onserverclick="btnRegister_ServerClick"
   validationgroup="RegisterGroup" >
```

In the code-behind we have added a method (groupIsValid) that will perform that validation on the group of validators defined by the passed parameter and return True if all validation for the group is valid, as shown in Examples 3-17 (VB) and 3-18 (C#).

The form-submit button event handlers will also be modified. Instead of testing the Page.IsValid property, they call the groupIsValid method with the applicable group name to determine if the data for the associated controls is valid.

> When the validation provided by ASP.NET is disabled, as described in this recipe, accessing the Page.IsValid property will throw an exception indicating that Page.IsValid cannot be called before validation has taken place. Page.IsValid cannot be accessed unless Page.Validate is called either by ASP.NET or in your code.

Example 3-16. Form with programmatic validation (.aspx)

```
<%@ Page Language="VB"
    MasterPageFile="~/ASPNetCookbookVB.master"
    AutoEventWireup="false"
    CodeFile="CH03ProgrammaticValidationVB.aspx.vb"
      Inherits="ASPNetCookbook.VBExamples.CH03ProgrammaticValidationVB"
    title="Programmatic Validation" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
    Programmatic Validation (VB)
 </div>
 <table align="center" class="dataEntry">
    </tr>
    <td colspan="2" align="left">
        <asp:ValidationSummary id="vsLoginErrors" Runat="server"
        CssClass="alertText"
        DisplayMode="BulletList"
        EnableClientScript="False"
        HeaderText="Error Summary"
        ValidationGroup="LoginGroup" />
  <asp:CustomValidator id="cvAuthentication" Runat="server"
       Display="None"
       EnableClientScript="False"
       ErrorMessage="Login ID or Password Is Invalid"
         OnServerValidate="cvAuthentication_ServerValidate"
       ValidationGroup="LoginGroup" />
  </td>
 </tr>
```

```
</tr>
 <td class="LabelText">Login ID: </td>
 <td>
    <asp:TextBox id="txtLoginID" Runat="server"
      Columns="30" CssClass="LabelText" />
    <asp:RequiredFieldValidator id="rfvLoginID"
   Runat="server"
   ControlToValidate="txtLoginID"
   CssClass="alertText"

   Display="Dynamic"
   EnableClientScript="False"
   ErrorMessage="Login ID Is Required"
   ValidationGroup="LoginGroup" >
    <img src="images/arrow_alert.gif" alt="arrow"/>
    </asp:RequiredFieldValidator>
 </td>
</tr>
</tr>
 <td class="LabelText">Password: </td>
 <td>
    <asp:TextBox id="txtPassword" Runat="server"
        TextMode="Password"
        Columns="30" CssClass="LabelText" />
    <asp:RequiredFieldValidator id="rfvPassword"
   Runat="server"
   ControlToValidate="txtPassword"
   CssClass="alertText"
   Display="Dynamic"
   EnableClientScript="False"
   ErrorMessage="Password Is Required"
   ValidationGroup="LoginGroup" >
   <img src="images/arrow_alert.gif" alt="arrow"/>
   </asp:RequiredFieldValidator>
 </td>
</tr>
</tr>
 <td colspan="2" align="center">
 <br/>
   <input id="btnLogin" runat="server" type="button"
   value="Login" causesvalidation="false"
   onserverclick="btnLogin_Click"
   validationgroup="LoginGroup" />
 </td>
</tr>
 </table>
 <hr />
 <table align="center" class="dataEntry">
</tr>
    <td colspan="2" align="left">
   <asp:ValidationSummary id="vsRegistrationErrors" Runat="server"
     CssClass="alertText"
```

```
      DisplayMode="BulletList"
      EnableClientScript="False"
      HeaderText="Error Summary"
      ValidationGroup="RegisterGroup" />
    </td>
 </tr>
 </tr>
    <td class="LabelText">Login ID: </td>
    <td>

       <asp:TextBox id="txtNewLoginID" Runat="server"
     Columns="30" CssClass="LabelText" />
       <asp:RequiredFieldValidator id="rfvNewLoginID"
        Runat="server"
    ControlToValidate="txtNewLoginID"
    CssClass="alertText"
    Display="Dynamic"
    EnableClientScript="False"
    ErrorMessage="Login ID Is Required"
    ValidationGroup="RegisterGroup" >
    <img src="images/arrow_alert.gif" alt="arrow"/>
    </asp:RequiredFieldValidator>
    </td>
 </tr>
 </tr>
    <td class="LabelText">Email Address: </td>
    <td>
    <asp:TextBox id="txtEmailAddress" Runat="server"
     Columns="30" CssClass="LabelText" />
       <asp:requiredfieldvalidator id="rfvEmailAddress"
    runat="server"
    controltovalidate="txtEmailAddress"
    cssclass="alertText"
    display="Dynamic"
    enableclientscript="False"
    ErrorMessage="Email Address Is Required"
    ValidationGroup="RegisterGroup" >
    <img src="images/arrow_alert.gif" alt="arrow"/>
       </asp:requiredfieldvalidator>
       <asp:RegularExpressionValidator id="revEmailAddress"
        Runat="server"
        ControlToValidate="txtEmailAddress"
        CssClass="alertText"
        Display="Dynamic"
        EnableClientScript="False"
ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
        ErrorMessage="Invalid Email Address"
    ValidationGroup="RegisterGroup" >
      <img src="images/arrow_alert.gif" alt="arrow"/>
       </asp:RegularExpressionValidator>
    </td>
 </tr>
```

```
</tr>
 <td class="LabelText">Password: </td>
 <td>
    <asp:TextBox id="txtPassword1" Runat="server"
   TextMode="Password"
    Columns="30" CssClass="LabelText" />
        <asp:RequiredFieldValidator id="rfvPassword1"
   Runat="server"
    ControlToValidate="txtPassword1"
   CssClass="alertText"

   Display="Dynamic"
   EnableClientScript="False"
   ErrorMessage="Password Is Required"
   ValidationGroup="RegisterGroup" >
      <img src="images/arrow_alert.gif" alt="arrow"/>
      </asp:RequiredFieldValidator>
 </td>
</tr>
</tr>
 <td class="LabelText">Re-enter Password: </td>
 <td>
       <asp:TextBox id="txtPassword2" Runat="server"
    TextMode="Password"
     Columns="30" CssClass="LabelText" />
    <asp:RequiredFieldValidator id="rvPassword2"
   Runat="server"
    ControlToValidate="txtPassword2"
   CssClass="alertText"
   Display="Dynamic"
   EnableClientScript="False"
   ErrorMessage="Re-Entered Password Is Required"
   ValidationGroup="RegisterGroup" >
       <img src="images/arrow_alert.gif" alt="arrow"/>
   </asp:RequiredFieldValidator>
       <asp:CompareValidator ID="cvPassword2" runat="server"
    ControlToValidate="txtPassword2"
    ControlToCompare="txtPassword1"
   CssClass="alertText"
   Display="Dynamic"
   EnableClientScript="False"
   ErrorMessage="Both Passwords Must Match"
   ValidationGroup="RegisterGroup" >
        <img src="images/arrow_alert.gif" alt="arrow"/>
   </asp:CompareValidator>
 </td>
</tr>
</tr>
 <td colspan="2" align="center">
   <br/>
   <input id="btnRegister" runat="server" type="button"
   value="Register" causesvalidation="false"
```

```
    onserverclick="btnRegister_ServerClick"
    validationgroup="RegisterGroup" />
  </td>
  </tr>
  </table>
</asp:Content>
```

Example 3-17. Form with programmatic validation (.vb)

```
Option Explicit On
Option Strict On
Imports System.Configuration.ConfigurationManager
Imports System.Data
Imports System.Data.OleDb
Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
  ''' CH03ProgrammaticValidationVB.aspx
  ''' </summary>
  Partial Class CH03ProgrammaticValidationVB
 Inherits System.Web.UI.Page
 '''**********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the authentication server
 ''' validate event. It is responsible for checking the login ID and
 ''' password in the database to authenticate the user.
 ''' </summary>
 '''
 ''' <param name="source">Set to the sender of the event</param>
 ''' <param name="args">Set to the event arguments</param>
 Protected Sub cvAuthentication_ServerValidate(ByVal source As Object, _
         ByVal args As ServerValidateEventArgs)
   Dim dbConn As OleDbConnection = Nothing
   Dim dCmd As OleDbCommand = Nothing
   Dim strConnection As String
   Dim strSQL As String
   Try
  'initially assume credentials are invalid
  args.IsValid = False
     'get the connection string from web.config and open a connection
  'to the database
  strConnection = _
   ConnectionStrings("dbConnectionString").ConnectionString
  dbConn = New OleDb.OleDbConnection(strConnection)
  dbConn.Open( )
  'build the query string and check to see if a user with the entered
  'credentials exists in the database
```

```
strSQL = "SELECT AppUserID FROM AppUser " & _
    "WHERE LoginID=? AND " & _
     "Password=?"
   dCmd = New OleDbCommand(strSQL, dbConn)
dCmd.Parameters.Add(New OleDbParameter("LoginID", _
          txtLoginID.Text))
dCmd.Parameters.Add(New OleDbParameter("Password", _
          txtPassword.Text))

'check to see if the user was found
If (Not IsNothing(dCmd.ExecuteScalar( ))) Then
 args.IsValid = True
End If
Finally
'cleanup
If (Not IsNothing(dbConn)) Then
 dbConn.Close( )
End If
End Try
 End Sub 'cvAuthentication_ServerValidate
 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the login button click
 ''' event. It is responsible for processing the form data for login.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnLogin_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
  'check to see if all entered login data is valid
  If (groupIsValid("LoginGroup")) Then
 'user has been authenticated so proceed with allowing access
 'to the site
  End If
 End Sub 'btnLogin_Click
 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the login button click
 ''' event. It is responsible for processing the form data for
 ''' registration.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnRegister_ServerClick(ByVal sender As Object, _
        ByVal e As System.EventArgs)
 'check to see if all entered registration data is valid
 If (groupIsValid("RegisterGroup")) Then
  'all entered data is valid so proceed with registration
 End If
 End Sub 'btnRegister_ServerClick
```

```
    '''**************************************************************************
    ''' <summary>
    ''' This routine iterates through validators for the passed group
    ''' performing the validation.
    ''' </summary>
    '''

    ''' <param name="validatorGroup">Set to the name of the validator
    ''' group to perform validation on.
    ''' </param>
    '''
    ''' <returns>True if all validators in the group are valid.
    '''         Else, False.
    '''</returns>
    Private Function groupIsValid(ByVal validatorGroup As String) As Boolean
   Dim validators As System.Web.UI.ValidatorCollection = Nothing
   Dim validator As System.Web.UI.IValidator = Nothing
   Dim isValid As Boolean = True
   'get the validators in the Register group
   validators = Page.GetValidators(validatorGroup)
   'iterate through the validators calling the Validate methods
   'and checking to see if the validation was succcessful
   For Each validator In validators
     validator.Validate( )
     If (Not validator.IsValid) Then
       isValid = False
     End If
   Next validator
   Return (isValid)
  End Function 'groupIsValid
   End Class 'CH03ProgrammaticValidationVB
End Namespace
```

Example 3-18. Form with programmatic validation (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
   ///  CH03ProgrammaticValidationCS.aspx
  /// </summary>
   public partial class CH03ProgrammaticValidationCS : System.Web.UI.Page
  {
 ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the authentication server
 /// validate event. It is responsible checking the login ID and password
 /// in the database to authenticate the user.
 /// </summary>
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>

 protected void cvAuthentication_ServerValidate(Object source,
    System.Web.UI.WebControls.ServerValidateEventArgs args)
 {
   OleDbConnection dbConn = null;
   OleDbCommand dCmd = null;
   String strConnection = null;
   String strSQL = null;
   try
   {
     // initially assume credentials are invalid
  args.IsValid = false;
  // get the connection string from web.config and open a connection
  // to the database
  strConnection = ConfigurationManager.
     ConnectionStrings["dbConnectionString"].ConnectionString;
  dbConn = new OleDbConnection(strConnection);
  dbConn.Open( );
  // build the query string and check to see if a user with the
  // entered credentials exists in the database
  strSQL = "SELECT AppUserID FROM AppUser " +
     "WHERE LoginID=? AND " +
     "Password=?";
  dCmd = new OleDbCommand(strSQL, dbConn);
  dCmd.Parameters.Add(new OleDbParameter("LoginID",
          txtLoginID.Text));
  dCmd.Parameters.Add(new OleDbParameter("Password",
          txtPassword.Text));
  // check to see if the user was found
  if (dCmd.ExecuteScalar( ) != null)
  {
     args.IsValid = true;
  }
   } // try
   finally
   {
  // cleanup
  if (dbConn != null)
  {
    dbConn.Close( );
  }
   } // finally
 } // cvAuthentication_ServerValidate
 ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the login button click
 /// event. It is responsible for providing access to the site for the
 /// user if authenticated.

 /// </summary>
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnLogin_Click(Object sender,
       System.EventArgs e)
 {
   // check to see if all entered login data is valid
   if (groupIsValid("LoginGroup"))
   {
  // user has been authenticated so proceed with allowing access
  // to the site
   }
 } //btnLogin_Click
 ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the register button click
 /// event. It is responsible for processing the form data for
 /// registration.
 /// </summary>
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnRegister_ServerClick(Object sender,
         System.EventArgs e)
 {
  // check to see if all entered registration data is valid
  if (groupIsValid("RegisterGroup"))
  {
    // all entered data is valid so proceed with registration
  }
 } //btnRegister_ServerClick
 ///*********************************************************************
 /// <summary>
 /// This routine iterates through validators for the passed group
 /// performing the validation.
 /// </summary>
 /// <param name="validatorGroup">Set to the name of the validator
 /// group to perform validation on.
 /// </param>
 /// <returns>True if all validators in the group are valid.
 ///   Else, False.
 ///</returns>
 private Boolean groupIsValid(String validatorGroup)
 {
   ValidatorCollection validators = null;
   Boolean isValid = true;
```

```
    // get the validators in the Register group
    validators = Page.GetValidators(validatorGroup);

    // iterate through the validators calling the Validate methods
    // and checking to see if the validation was succcessful
    foreach (IValidator validator in validators)
    {
   validator.Validate( );
      if (!validator.IsValid)
   {
      isValid = false;
      }
   } // foreach validator
    return (isValid);
  } // groupIsValid
   } //  CH03ProgrammaticValidationCS
}
```

PREV

PREV

Chapter 4. Forms

PREV

PREV

4.0 Introduction

At the most basic level, forms provide the visual interface of your applications. Each form is a combination of programming logic and user interface rendered as an HTML page by the user's browser. ASP.NET server controls provide the basic building blocks of forms and expose an object model containing properties, methods, and events. Building a basic form in ASP.NET only requires adding some HTML and some server controls to a page and then handling some server control events in the code-behind to modify and interact with the page. These are the basics of forms and are generally well understood even by beginning ASP.NET programmers.

Beyond the basics, though, certain aspects of forms take some getting used to in ASP.NET: in particular, the concept of programming the server and client sides of an application. Forms are inherently client-side because that's where they execute and much of their behavior must be handlec in the client browser. This sounds simple enough, but getting forms to do your bidding requires you t write server-side code that writes the client-side code to be executed when the page is loaded in the browser. If that isn't enough, client-side code often needs to be written in JavaScript. Add to that the task of managing the nuances of another scripting language, and you begin to feel as though you're playing 3-D chess.

This chapter provides solutions to many form-related problems you are likely to encounter in using ASP.NET. By the time you've waded through a recipe or two, the required client- and server-side maneuvers ought to be manageable.

Microsoft has made several aspects of working with forms easier in ASP.NET 2.0. For instance, setting a form's default button is a common requirement for many web applications. In ASP.NET 1.x and in classic ASP, setting a form's default button requires you to write custom client-side JavaScript In ASP.NET 2.0, however, you can use the new `DefaultButton` property of the form object, as we explain in this chapter's first recipe.

Recipe 4.2. Setting the Default Button to Submit a Form

Problem

You have a form with multiple buttons and you need to set the button that will be used as the default button when the user presses the Enter key on the keyboard.

Solution

Create the *.aspx* file with the controls required for your application. In the `Page_Load` event handler of the code-behind class, use the .NET language of your choice to set the default button on the form.

Discussion

Setting the default button in ASP.NET 1.x and in classic ASP requires you to write custom client-side JavaScript that executes when the page is loaded in the browser. The script captures the `keypress` event, checks to see if the key that is pressed is the Enter key, and performs a `postback` of the form. In addition, the JavaScript created is typically different for each major browser.

ASP.NET 2.0, on the other hand, has exposed a `DefaultButton` property of the form object that allows you to set the button that will be used as the default when the user presses the Enter key. ASP.NET then handles the client script generation for you.

```
Page.Form.DefaultButton = btnLogin.UniqueID

Page.Form.DefaultButton = btnLogin.UniqueID;
```

Examples 4-1, 4-2, through 4-3 show the *.aspx* and code-behind files for an application that makes use of this solution.

When setting the DefaultButton property, note the following:

- The control that will be used as the default button when the user presses the Enter key can be any control that implements the IButtonControl interface.
- The DefaultButton property must be set to the UniqueID property of the desired control.

Failure to follow these guidelines results in an exception being thrown when the page is displayed, indicating the DefaultButton must be a control that implements IButtonControl.

See Also

Recipes 4.4 and 4.5

Example 4-1. Setting the default submit button (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH04SettingDefaultSubmitButtonVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH04SettingDefaultSubmitButtonVB"
  Title="Setting Default Submit Button" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Setting Default Submit Button (VB)
 </div>
 <table width="50%" align="center" border="0">
  <tr>
   <td class="labelText">Email Address: </td>
   <td>
    <asp:TextBox ID="txtEmailAddress" Runat="server"
      Columns="30" CssClass="LabelText" />
   </td>
   </tr>
   <tr>
   <td class="labelText">Password: </td>
   <td>
    <asp:TextBox ID="txtPassword" Runat="server"
     textmode="Password"
      Columns="30" CssClass="LabelText" />
    </td>
     </tr>
     <tr>
    <td colspan="2">
```

```
       <br/>
        <table align="center" width="50%">
    <tr>
       <td align="center">
        <asp:Button ID="btnLogin" runat="server"
       OnClick="btnLogin_Click"
       Text="Login" />
    </td>
    <td align="center">
      <asp:Button ID="btnCancel" runat="server"
        OnClick="btnCancel_Click"
        Text="Cancel" />
    </td>
    </tr>
    </table>
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 4-2. Setting the default submit button (.vb)

```
Option Explicit On
Option Strict On
Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
   '''  CH04SettingDefaultSubmitButtonVB.aspx
  ''' </summary>

  Partial Class CH04SettingDefaultSubmitButtonVB
    Inherits System.Web.UI.Page

 '''*********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
   'set the button that will be the default button when the user hits
   'the Enter key. NOTE: The UniqueID must be used or error will occur
   'when page is displayed indicating a control that implements
   'IButtonControl must be used as the default button
```

```
    Page.Form.DefaultButton = btnLogin.UniqueID
     End Sub 'Page_Load

   '''*************************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the login button click
   ''' event. It is responsible for processing the form data.
   ''' </summary>
   '''
   ''' <param name="sender">Set to the sender of the event</param>
   ''' <param name="e">Set to the event arguments</param>
   Protected Sub btnLogin_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)
  'perform login operations here
   End Sub 'btnLogin_Click

  '''**************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the cancel button click
  ''' event.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnCancel_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)
    'perform cancel operations here
  End Sub 'btnCancel_Click
  End Class
End Namespace
```

Example 4-3. Setting the default submit button (.cs)

```
using System;
namespace ASPNetCookbook.CSExamples
{
   /// <summary>
   /// This class provides the code behind for
    /// CH04SettingDefaultSubmitButtonCS.aspx
   /// </summary>
    public partial class CH04SettingDefaultSubmitButtonCS : System.Web.UI.Page
    {
     ///*****************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
```

```
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
  // set the button that will be the default button when the user hits
  // the Enter key. NOTE: The UniqueID must be used or error will occur
  // when page is displayed indicating a control that implements
  // IButtonControl must be used as the default button
  Page.Form.DefaultButton = btnLogin.UniqueID;
 } // Page_Load

 ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the login button click
 /// event. It is responsible for processing the form data.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnLogin_Click(Object sender,
     System.EventArgs e)
   {
  // perform login operations here
   }  //btnLogin_Click

   ///*********************************************************************
   /// <summary>
   /// This routine provides the event handler for the cancel button click
   /// event.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void btnCancel_Click(Object sender,
     System.EventArgs e)
   {
  // perform cancel operations here
   }   //btnCancel_Click
  } // CH04SettingDefaultSubmitButtonCS
}
```

Recipe 4.3. Submitting a Form to a Different Page

Problem

You need to submit the information on one pagea form, for exampleto another. You might want to do this to use one page to collect form data and a second page to process it.

The default operation for ASP.NET is to submit the form to the same page. Submitting the form to a different pagei.e., performing *cross-page posting* requires some coding gymnastics in ASP.NET 1.x, explained in some detail in the first edition of the *ASP.NET Cookbook*:

- Using the `Server.Transfer` method in a button click event handler in the code-behind of the first page to transfer control to a second page, after saving the contents of the first in session scope.
- Using the `Server.Transfer` method in a button click event handler in the code-behind of the first page to transfer control, along with the form contents in the `ViewState`, to the second page.

In ASP.NET 2.0, however, performing cross-page posting is easier because you can now accomplish it by setting the `PostBackUrl` button property.

Solution

Use the cross-page posting functionality added to ASP.NET 2.0 to submit a form to a different page.

In the *.aspx* file of the first page, set to the URL of the desired page the `PostBackUrl` property of the button that will initiate the submission of the form to another page. No special code is required in the code-behind of the first page to support cross-page posting.

In the code-behind of the second page, use the .NET language of your choice to:

1. Check to verify that page is being requested as a cross-page post from another page.
2. Use the `FindControl` method of the `PreviousPage` property to get a reference to a control in the first page.
3. Use the data from the control in the first page as required in the second page.
4. Repeat the last two steps for each control in the first page that you need data for in the second

page.

Examples 4-4, 4-5, 4-6, 4-7, 4-8 through 4-9 showthe *.aspx* and code-behind files for an application that implements this solution.

Discussion

The default operation for ASP.NET is to submit the form to the same page for processing. This operation is acceptable for most applications; however, the need may arise to submit the form to another page. This is much easier to do in ASP.NET 2.0 than in previous versions. In fact, prior to ASP.NET 2.0, submitting a form to another page required working around the system instead of with it.

In ASP.NET 2.0, the submission to another page is controlled by the`PostBackURL` property of the buttons on the form. This provides the ability to submit the formto different pages for each button.

```
<asp:Button ID="btnSubmit" runat="server"
  Text="Submit"
  PostBackUrl="~/CH04SubmitToAnother_SecondPageVB1.aspx" />
```

You cannot set the action attribute of the form element to cause the form to be submitted to another page. ASP.NET always changes the action to the URL of the page being displayed.

When the user clicks the submit button, ASP.NET executes a client-side JavaScript method that performs the submission to the second page, passing the information necessary for the second page to determine if the request is the result of a cross-page post, as well as the data to rehydrate the page object for the first page.

You can determine if a page is being requested as a result of a cross-page post by checking if the `PreviousPage` property of the Page object is `null` (`Nothing` in VB):

```
If (IsNothing(Page.PreviousPage)) Then
 'page is not being accessed by a cross-page postback

Else
 'page is being accessed by a cross-page postback

End If

if (Page.PreviousPage == null)
{
  // page is not being accessed by a cross-page postback
}
```

```
else
{
  // page is being accessed by a cross-page post-back
}
```

The first time you access the `PreviousPage` property, ASP.NET rehydrates the page object for the first page. As part of the rehydration, all events upto `LoadComplete` are fired. This includes the button click event for the submit button that was clicked. If your code contains an event handler for the click event for the buttons used to submit to another page, this code will be executed.

By accessing the `PreviousPage` property, your code can access the content of the controls on the first page. Since the `PreviousPage` property is of type `Page`, you cannot directly access the controls. You have to use the `FindControls` method to get a reference to the desired control and then access the data in the control.

```
get the page content control
'NOTE: This is required since a master page is being used and the
'    controls that contain the data needed here are within it
pageContent = CType(Page.PreviousPage.Form.FindControl("PageBody"), _
   ContentPlaceHolder)
'get the first name data from the first page and set the label
'on this page
tBox = CType(pageContent.FindControl("txtFirstName"), _
   TextBox)
lblFirstName.Text = tBox.Text
```

```
// get the page content control
// NOTE: This is required since a master page is being used and the
//   controls that contain the data needed here are within it
pageContent = (ContentPlaceHolder)
     (Page.PreviousPage.Form.FindControl("PageBody"));

  // get the first name data from the first page and set the label
  // on this page
  tBox = (TextBox)(pageContent.FindControl("txtFirstName"));
  lblFirstName.Text = tBox.Text;
```

If the control you are accessing is contained within another control, such as a template or a content placeholder, you need to use`FindControl` to get a reference to the container first and then use`FindControl` again to get a reference to the control containing your data.

Though the solution shown above works fine and is acceptable in most applications, it has two problems. First, you must have knowledge of the structure of the first pageto get a reference to the desired controls in the second (described above). This can become complicated if the controls on the first page are embedded in multiple layers of control containers. Second, the access to the controls is not strongly typed, requiring you to cast the control type for access. If the control type changes on the first page, the code in the second page must also be changed.

Another solution that eliminates these problems is to add public properties to the code-behind of the first page that expose the data in the controls that you need in the second page:

VB

```
'''*****************************************************************
''' <summary>
''' This routine provides the ability to get/set the firstName property
''' </summary>
Public Property firstName() As String
 Get
   Return (txtFirstName.Text)
 End Get
 Set(ByVal value As String)
  txtFirstName.Text = value
 End Set
End Property

'''*****************************************************************
''' <summary>
''' This routine provides the ability to get/set the lastName property
''' </summary>
Public Property lastName() As String
 Get
    Return (txtLastName.Text)
 End Get
 Set(ByVal value As String)
txtLastName.Text = value
 End Set
End Property

'''****************************************************************
 ''' <summary>
 ''' This routine provides the ability to get/set the age property
 ''' </summary>
 Public Property age() As String
 Get
   Return (txtAge.Text)
 End Get
 Set(ByVal value As String)
txtAge.Text = value
   End Set
 End Property
```

```
///**********************************************************************
/// <summary>
/// This routine provides the ability to get/set the firstName property
/// </summary>
public String firstName
{
  get
  {
  return (txtFirstName.Text);
  }
  set
  {
  txtFirstName.Text = value;
  }
 } // firstName

 ///*********************************************************************
 /// <summary>
 /// This routine provides the ability to get/set the lastName property
 /// </summary>
 public String lastName
 {
 get
 {
  return (txtLastName.Text);
 }
 set
 {
    txtLastName.Text = value;
 }
  } // lastName

 ///*********************************************************************
 /// <summary>
 /// This routine provides the ability to get/set the age property
 /// </summary>
 public String age
 {
 get
 {
  return (txtAge.Text);
 }
 set
 {
  txtAge.Text = value;
 }
} // age
```

In the *.aspx* file of the second page, add the `PreviousPageType` directive to the top of the file with the `VirtualPath` attribute set to the URL of the first page:

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH04SubmitToAnother_SecondPageVB2.aspx.vb"
  Inherits="ASPNetCookbook.VBExamples.CH04SubmitToAnother_SecondPageVB2"
  Title="Form Submission To Another Page - Second Page - Approach 2" %>
   <%@ PreviousPageType VirtualPath="~/CH04SubmitToAnother_FirstPageVB2.aspx" %>
```

References to pages, images, and other resources are typically prefaced with "~/", which is used to indicate the root folder of the web application.

In the code-behind of the second page, define a variable of the type of the first page and set its valu to the `PreviousPage` property of the `Page` object:

```
Dim prevPage As CH04SubmitToAnother_FirstPageVB2

'get a strongly type reference to the previous page
prevPage = CType(Page.PreviousPage, _
    CH04SubmitToAnother_FirstPageVB2)
```

```
CH04SubmitToAnother_FirstPageCS2  prevPage;

// get a strongly type reference to the previous page
prevPage  =  (CH04SubmitToAnother_FirstPageCS2)(Page.PreviousPage);
```

With the strongly typed reference to the first page, you can directly access the properties you added to access the required data:

```
lblFirstName.Text = prevPage.firstName
lblLastName.Text = prevPage.lastName
lblAge.Text = prevPage.age
```

```
lblFirstName.Text = prevPage.firstName;
lblLastName.Text = prevPage.lastName;
lblAge.Text = prevPage.age;
```

This second solution is cleaner and less likely to break when the code is changed during future maintenance operations. It does require a little more planning and coding but it is worth the effort.

See Also

Search *ASP.NET 2.0 Internals* in the MSDN Library.

Example 4-4. Submitting a form to another pagefirst page (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH04SubmitToAnother_FirstPageVB1.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH04SubmitToAnother_FirstPageVB1"
  Title="Form Submission To Another Page - Approach 1" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
   <div align="center" class="pageHeading">
      Form Submission To Another Page - Approach 1 (VB)
   </div>
    <table width="50%" align="center" border="0">
   <tr>
     <td class="labelText">First Name: </td>
  <td>
     <asp:TextBox ID="txtFirstName" Runat="server"
     Columns="30" CssClass="LabelText" />
 </td>
  </tr>
  <tr>
     <td class="labelText">Last Name: </td>
  <td>
  <asp:TextBox ID="txtLastName" Runat="server"
     Columns="30" CssClass="LabelText" />
 </td>
  </tr>
  <tr>
  <td class="labelText">Age: </td>
  <td>
    <asp:TextBox ID="txtAge" Runat="server"
     Columns="30" CssClass="LabelText" />
 </td>
  </tr>
  <tr>
     <td align="center" colspan="2">
    <br/>
    <asp:Button ID="btnSubmit" runat="server"
      Text="Submit"
      PostBackUrl="~/CH04SubmitToAnother_SecondPageVB1.aspx" />

   </td>
 </tr>
  </table>
</asp:Content>
```

Example 4-5. Submitting a form to another pagefirst page (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH04SubmitToAnother_FirstPageVB1.aspx
 ''' </summary>
 Partial  Class  CH04SubmitToAnother_FirstPageVB1
  Inherits System.Web.UI.Page
   '''*****************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Load(ByVal sender As Object, _
       ByVal e As System.EventArgs) Handles Me.Load
     End Sub 'Page_Load
    End  Class  'CH04SubmitToAnother_FirstPageVB1
End Namespace
```

Example 4-6. Submitting a form to another pagefirst page (.cs)

```
using System;
namespace ASPNetCookbook.CSExamples
{
   /// <summary>
   /// This class provides the code behind for
    /// CH04SubmitToAnother_FirstPageCS1.aspx
   /// </summary>
    public partial class CH04SubmitToAnother_FirstPageCS1 : System.Web.UI.Page
   {
     ///**********************************************************************
 /// <summary>
    /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
    {
 } // Page_Load
  } // CH04SubmitToAnother_FirstPageCS1
}
```

Example 4-7. Submitting a form to another pagesecond page (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
   AutoEventWireup="false"
     CodeFile="CH04SubmitToAnother_SecondPageVB1.aspx.vb"
     Inherits="ASPNetCookbook.VBExamples.CH04SubmitToAnother_SecondPageVB1"
   Title="Form Submission To Another Page - Second Page - Approach 1" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
    <div align="center" class="pageHeading">
     Form Submission To Another Page - Approach 1 (VB)
 </div>
 <table width="50%" align="center" border="0">
   <tr>
      <td colspan="2" align="center" class="pageHeading">
    Data Submitted From Previous Page
  </td>
   </tr>
   <tr>
      <td class="labelText">First Name: </td>
   <td class="labelText">
   <asp:Label ID="lblFirstName" Runat="server" />
   </td>
```

```
    </tr>
    <tr>
    <td class="labelText">Last Name: </td>
    <td class="labelText">
     <asp:Label id="lblLastName" Runat="server" />
    </td>
   </tr>
   <tr>
    <td class="labelText">Age: </td>
    <td class="labelText">
     <asp:Label ID="lblAge" Runat="server" />
    </td>
   </tr>
   <tr>
    <td colspan="2">&nbsp;</td>
   </tr>
   <tr>
       <td class="labelText">Page Access Method: </td>
    <td class="labelText">
     <asp:Label ID="lblPageAccessMethod" Runat="server" />
    </td>
   </tr>
  </table>
</asp:Content>
```

Example 4-8. Submitting a form to another pagesecond page (.vb)

```
Option Explicit On
Option Strict On

Imports System.Web.UI.WebControls

Namespace ASPNetCookbook.VBExamples
   ''' <summary>
   ''' This class provides the code behind for
    ''' CH04SubmitToAnother_SecondPageVB1.aspx
    ''' </summary> Partial Class CH04SubmitToAnother_SecondPageVB1
  Inherits System.Web.UI.Page
   '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
   Dim tBox As TextBox
   Dim pageContent As ContentPlaceHolder
   'check to see how the page is being accessed
   If (IsNothing(Page.PreviousPage)) Then
   'page is not being accessed by cross-page post so check to if
   'it is being accessed via self post-back
   If (Page.IsPostBack) Then
    'page is being accessed by a post-back from itself
    lblPageAccessMethod.Text = "Page was accessed via post-back"
   Else
    'page is being accessed directly
    lblPageAccessMethod.Text = "Page was accessed directly"
   End If
  Else
   'page is being accessed by a cross-page post-back
   lblPageAccessMethod.Text = "Page was accessed via cross-page post-back"

   'get the page content control
   'NOTE: This is required since a master page is being used and the
   '        controls that contain the data needed here are within it
   pageContent = CType(Page.PreviousPage.Form.FindControl("PageBody"), _
      ContentPlaceHolder)

   'get the first name data from the first page and set the label
   'on this page
   tBox = CType(pageContent.FindControl("txtFirstName"), _
      TextBox)
   lblFirstName.Text = tBox.Text

   'get the last name data from the first page and set the label
   'on this page
   tBox = CType(pageContent.FindControl("txtLastName"), _
     TextBox)
   lblLastName.Text = tBox.Text

   'get the age data from the first page and set the label
   'on this page
   tBox = CType(pageContent.FindControl("txtAge"), _
    TextBox)
   lblAge.Text = tBox.Text
    End If
  End Sub 'Page_Load
 End  Class  'CH04SubmitToAnother_SecondPageVB1
End Namespace
```

Example 4-9. Submitting a form to another pagesecond page (.cs)

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

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
   ///  CH04SubmitToAnother_SecondPageCS1.aspx
 /// </summary>
 public partial class CH04SubmitToAnother_SecondPageCS1 : System.Web.UI.Page
 {
 ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
  TextBox tBox;
  ContentPlaceHolder pageContent;

  // check to see how the page is being accessed
  if (Page.PreviousPage == null)
  {
     // page is not being accessed by cross-page post so check to if
  // it is being accessed via self post-back
  if (Page.IsPostBack)
  {
    // page is being accessed by a post-back from itself
    lblPageAccessMethod.Text = "Page was accessed via post-back";
  }
  else
  {
    // page is being accessed directly
    lblPageAccessMethod.Text = "Page was accessed directly";
        }
   }
   else
   {
      // page is being accessed by a cross-page post-back
   lblPageAccessMethod.Text = "Page was accessed via cross-page post-back";

   // get the page content control
   // NOTE: This is required since a master page is being used and the
   //     controls that contain the data needed here are within it
   pageContent = (ContentPlaceHolder)
       (Page.PreviousPage.Form.FindControl("PageBody"));

   // get the first name data from the first page and set the label
```

```
    // on this page
    tBox = (TextBox)(pageContent.FindControl("txtFirstName"));
    lblFirstName.Text = tBox.Text;

    // get the last name data from the first page and set the label
    // on this page
    tBox = (TextBox)(pageContent.FindControl("txtLastName"));
    lblLastName.Text = tBox.Text;

    // get the age data from the first page and set the label
    // on this page
    tBox = (TextBox)(pageContent.FindControl("txtAge"));
    lblAge.Text = tBox.Text;
    }
  }  // Page_Load
   }  //  CH04SubmitToAnother_SecondPageCS1
}
```

PREV

Recipe 4.4. Simulating Multipage Forms Problem

Problem

You want to create a form that appears, from the user's perspective, to consist of multiple pages, while keeping all of your code in one *.aspx* file and the code-behind that accompanies it.

Solution

Create one ASP.NET page. Use a `Wizard` control with a `WizardStep` control containing the HTML for each the "virtual" pages you wish to display. The output of a typical multipage form is shown in Figures 4-1, 4- 4-3 , 4-4 through 4-5 . Examples 4-10 , 4-11 through 4-12 show the *.aspx* and code-behind files for an application that implements this solution.

Figure 4-1. Multipage form output (Section 1.1)

Discussion

In classic ASP, a series of questions or prompts, such as those on a survey or wizard, is typically implemented using multiple ASP pages with each submitting to the next in turn. Because ASP.NET 2.0 a you to submit a form to another page, this same approach can still be used; however, ASP.NET 2.0 provides a simpler approach using the `Wizard` control.

Figure 4-2. Multipage form output (Section 1.2)

Figure 4-3. Multipage form output (Section 1.2.3)

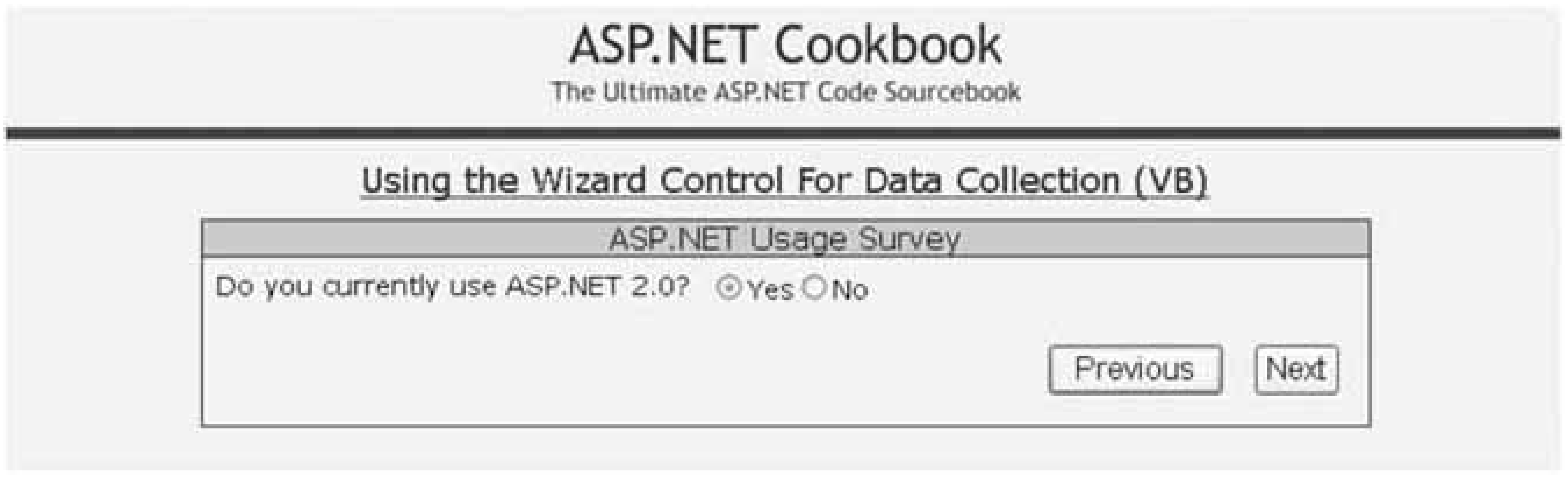

ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Using the Wizard Control For Data Collection (VB)
ASP.NET Usage Survey
Do you currently use ASP.NET 2.0? [✓ Yes] No

Figure 4-4. Multipage form output (Section 1.2.4)

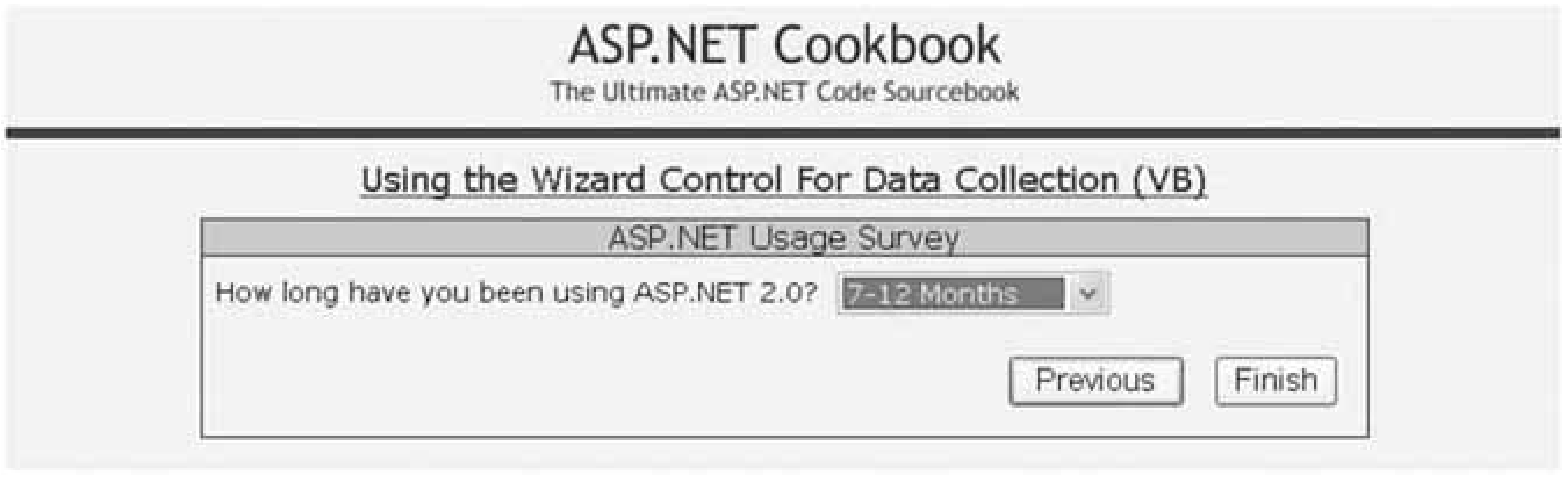

ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Using the Wizard Control For Data Collection (VB)
ASP.NET Usage Survey
How long have you been using ASP.NET 2.0? 7-12 Months

Figure 4-5. Multipage form output (Section 1.3)

The `Wizard` control provides the infrastructure you need to present a series of virtual pages to the user including the requisite navigation controls for moving forward and backward in the series. The virtual pa are defined using a `WizardStep` control for each of the individual pages you want to present to the user. defining the step type for the `WizardStep` controls (`Start, Step, Finish, Complete` , and `Auto`), the `Wiza` control will display the appropriate navigation buttons. The navigation buttons displayed for each step ty are shown below.

Table 4-1.

WizardStep type	Navigation buttons displayed
Start	Next
Step	Previous, Next
Finish	Previous, Complete
Complete	None
Auto	Automatically generates the navigation buttons as required by the position of the WizardStep control in the collection of WizardSteps

The example solution we present here uses the Wizard control and a series of WizardStep controls to disp a series of questions in a short survey.

Refer to Recipe 4.2 if you are determined to stick to the multiple form approach.

In our example, the *.aspx* file contains one Wizard control and five WizardStep controls. The first WizardS contains the first question ("Do you currently use ASP.NET 1.x?") along with a RadioButtonList control fo the response. The second WizardStep control contains the second question ("How long have you been us ASP.NET 1.x?") along with a DropDownList control for the response. The third and fourth WizardStep controls are the same as the first and second steps with questions related to ASP.NET 2.0. The fifth WizardStep control contains a message thanking the user for taking the survey.

The Page_Load event handler in the code-behind initializes two arrays containing the possible answers to questions and binds the data to the RadioButtonList and DropDownList controls in steps 1 through 4, as shown in Examples 4-11 (VB) and 4-12 (C#). We used this approach to reuse the data and to set the sta for potentially populating the available responses from a database.

We have implemented event handlers in the code-behind for the wizard's next and previous button click events to demonstrate the ability to skip questions as a function of the user's responses. In the event handler for the next button click event (wzSurvey_NextButtonClick), we check to see if the current step for one of the "use" questions. If the user responds by indicating she has not used ASP.NET, there is no point in asking how long she has used the product, so we skip the next step using the wizard's MoveTo method.

```
'skip steps as a function of the users answers
Select Case e.CurrentStepIndex
 Case 0
  If ((rbStep1.SelectedIndex >= 0) AndAlso _
    (rbStep1.SelectedItem.Text.Equals("No"))) Then
  'user does not use ASP.NET 1.x so move to 2.0 question
  wzSurvey.MoveTo(step3)
```

```
End If

 Case 2
   If ((rbStep1.SelectedIndex >= 0) AndAlso _
     (rbStep3.SelectedItem.Text.Equals("No"))) Then
  'user does not use ASP.NET 2.0 so move to complete step
  wzSurvey.MoveTo(step5)
   End If

Case Else
  'nothing required
 End Select
```

C#

```
 // skip steps as a function of the users answers
 switch (e.CurrentStepIndex)
 {
case 0:
  if ((rbStep1.SelectedIndex >= 0) &&
       (rbStep1.SelectedItem.Text.Equals("No")))
  {
    // user does not use ASP.NET 1.x so move to 2.0 question
 wzSurvey.MoveTo(step3);
  }
  break;

  case 2:
 if ((rbStep1.SelectedIndex >= 0) &&
      (rbStep3.SelectedItem.Text.Equals("No")))
 {
   // user does not use ASP.NET 2.0 so move to complete step
wzSurvey.MoveTo(step5);
 }
   break;

 default:
   // nothing required
break;

  }
```

In the wizard's previous button click event handler (`wzSurvey_PreviousButtonClick`), we do the same checks, but this time for the user navigating in the reverse direction.

When the final step is reached, the survey should be saved in your data store. We do this in the event handler for the `ActiveStepChanged` event by checking to see if the `ActiveStepIndex` is equal to the last st in the series.

```
'check to see if the complete step is the active step
If (wzSurvey.ActiveStepIndex = wzSurvey.WizardSteps.Count - 1) Then
 'survey is complete so production application should store the data
 'in an applicable data store
End If
```

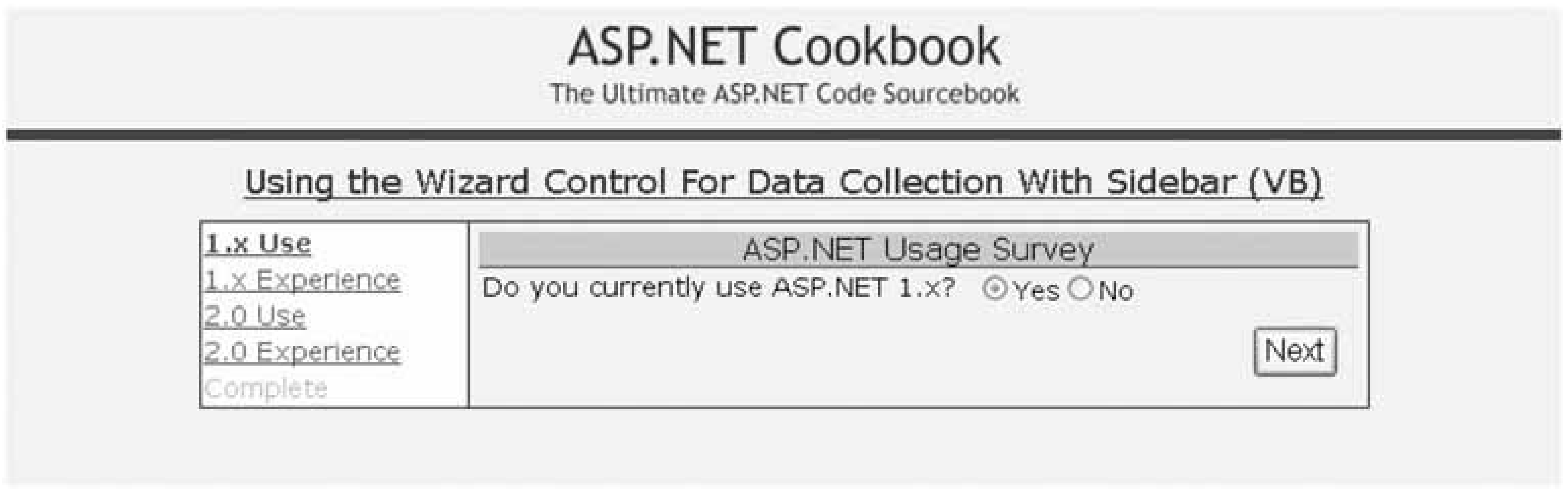
```
// check to see if the complete step is the active step
if (wzSurvey.ActiveStepIndex == wzSurvey.WizardSteps.Count - 1)
{
  // survey is complete so production application should store the data
  // in an applicable data store
}
```

The `Wizard` control has a `Finish` button click event that can be used to trigger your code to save the results; however, if your implementation of the `Wizard` control (like our example) allows the user to skip steps, it is possible the Finish step will never be displayed and the `Finish` button click event will not occur.

In addition to sequential navigation, the `Wizard` control provides the ability to perform direct navigation through the series of steps by displaying a sidebar with links to each step, as shown in Figure 4-6 The sidebar is enabled by setting the `DisplaySideBar` attribute of the `Wizard` control to true.

```
<asp:Wizard ID="wzSurvey" runat="server"
       CssClass="wizardBody" Width="75%" align="center"
      HeaderText="ASP.NET Usage Survey"
      HeaderStyle-CssClass="wizardHeader"
      StepStyle-HorizontalAlign="Left"
      StepStyle-VerticalAlign="Middle"
      DisplaySideBar="true"
      SideBarStyle-CssClass="wizardSideBar"
      OnNextButtonClick="wzSurvey_NextButtonClick"
      OnPreviousButtonClick="wzSurvey_PreviousButtonClick"
      OnActiveStepChanged="wzSurvey_ActiveStepChanged" >
```

Figure 4-6. Wizard control with sidebar navigation


ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

Using the Wizard Control For Data Collection With Sidebar (VB)

1.x Use
1.x Experience
2.0 Use
2.0 Experience
Complete

ASP.NET Usage Survey

Do you currently use ASP.NET 1.x? [✓ Yes] No

Next

By using the style attributes for each of the sections of the`Wizard` control, almost everything about the `Wizard` control can be configured to match the look and feel of your application.

See Also

Recipe 4.2

Example 4-10. Simulating a multipage form (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH04SurveyDataVB1.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH04SurveyDataVB1"
 Title="Using the Wizard Control For Data Collection" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">
  Using the Wizard Control For Data Collection (VB)
  </div>
  <asp:Wizard ID="wzSurvey" runat="server"
      CssClass="wizardBody" Width="75%" align="center"
      HeaderText="ASP.NET Usage Survey"
      HeaderStyle-CssClass="wizardHeader"
      StepStyle-HorizontalAlign="Left"
      StepStyle-VerticalAlign="Middle"
      DisplaySideBar="false"
      OnNextButtonClick="wzSurvey_NextButtonClick"
      OnPreviousButtonClick="wzSurvey_PreviousButtonClick"
      OnActiveStepChanged="wzSurvey_ActiveStepChanged" >
 <WizardSteps>
  <asp:WizardStep ID="step1" runat="server"
    StepType="Start" Title="1.x Use">
   <asp:Label ID="lbl1" runat="server"
    CssClass="wizardStep"
    Text="Do you currently use ASP.NET 1.x?" />
   <asp:RadioButtonList ID="rbStep1" Runat="server"
      RepeatLayout="Flow"
      RepeatDirection="Horizontal"
      CssClass="wizardStep" />
   </asp:WizardStep>

 <asp:WizardStep ID="step2" runat="server"
       StepType="Step" Title="1.x Experience">
   <asp:Label ID="lbl2" runat="server"
        CssClass="wizardStep"
    Text="How long have you been using ASP.NET 1.x?" />
   <asp:DropDownList ID="ddStep2" runat="server"
        CssClass="wizardStep" />
 </asp:WizardStep>

 <asp:WizardStep ID="step3" runat="server"
    StepType="Step" Title="2.0 Use">
 <asp:Label ID="lbl3" runat="server"
     CssClass="wizardStep"
     Text="Do you currently use ASP.NET 2.0?" />
 <asp:RadioButtonList ID="rbStep3" Runat="server"
        RepeatLayout="Flow"
    RepeatDirection="Horizontal"
    CssClass="wizardStep" />
 </asp:WizardStep>

 <asp:WizardStep ID="step4" runat="server"
       StepType="Finish" Title="2.0 Experience">
  <asp:Label ID="lbl4" runat="server"
      CssClass="wizardStep"
      Text="How long have you been using ASP.NET 2.0?" />
  <asp:DropDownList ID="ddStep4" runat="server"
     CssClass="wizardStep" />
 </asp:WizardStep>

 <asp:WizardStep ID="step5" runat="server"
    Ste1pType="Complete"
    Title="Complete" >
  <asp:Label ID="lbl5" runat="server"
    CssClass="wizardStep"
    Text="Thank you for taking our survey" />
  </asp:WizardStep>
 </WizardSteps>
  </asp:Wizard>
</asp:Content>
```

Example 4-11. Simulating a multipage form (.vb)

```
Option Explicit On
Option Strict On
```

```
Imports System.Web.UI.WebControls

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' CH04SurveyDataVB1.aspx
 ''' </summary>
 Partial Class CH04SurveyDataVB1
  Inherits System.Web.UI.Page

  '''***********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Load(ByVal sender As Object, _
         ByVal e As System.EventArgs) Handles Me.Load

  Dim yesNoSelections As ArrayList
  Dim experienceSelections As ArrayList

  If (Not Page.IsPostBack) Then
   'build the arraylist with the yes/no responses
   yesNoSelections = New ArrayList()
   yesNoSelections.Add(New ListItem("Yes", "1"))
   yesNoSelections.Add(New ListItem("No", "0"))

    'bind the yes/no data to the radio button lists in the questions
  rbStep1.DataSource = yesNoSelections
  rbStep1.DataTextField = "Text"
  rbStep1.DataValueField = "Value"
  rbStep1.DataBind()

  rbStep3.DataSource = yesNoSelections
  rbStep3.DataTextField = "Text"
  rbStep3.DataValueField = "Value"
  rbStep3.DataBind()

  'build the arraylist with the experience responses
  experienceSelections = New ArrayList
  experienceSelections.Add(New ListItem("-- Select One --", "0"))
  experienceSelections.Add(New ListItem("0-6 Months", "1"))
  experienceSelections.Add(New ListItem("7-12 Months", "2"))
  experienceSelections.Add(New ListItem("13-24 Months", "3"))
  experienceSelections.Add(New ListItem("24+ Months", "4"))

  'bind the experience data to the radio button lists in the questions
  ddStep2.DataSource = experienceSelections
  ddStep2.DataTextField = "Text"
  ddStep2.DataValueField = "Value"
  ddStep2.DataBind()

  ddStep4.DataSource = experienceSelections
  ddStep4.DataTextField = "Text"
  ddStep4.DataValueField = "Value"
  ddStep4.DataBind()
 End If
End Sub 'Page_Load

'''************************************************************************** ''' <summar
''' This routine provides the event handler for the wizard's next button
''' click event. It is responsible for altering the survey navigation as
''' as function of the answers provided
''' </summary>
'''
''' <param name="sender"></param>
''' <param name="e"></param>
Protected Sub wzSurvey_NextButtonClick(ByVal sender As Object, _
     ByVal e As WizardNavigationEventArgs)
   'skip steps as a function of the users answers
   Select Case e.CurrentStepIndex
  Case 0
     If ((rbStep1.SelectedIndex >= 0) AndAlso _
      (rbStep1.SelectedItem.Text.Equals("No"))) Then
    'user does not use ASP.NET 1.x so move to 2.0 question
   wzSurvey.MoveTo(step3)
     End If

     Case 2
        If ((rbStep1.SelectedIndex >= 0) AndAlso _
         (rbStep3.SelectedItem.Text.Equals("No"))) Then
    'user does not use ASP.NET 2.0 so move to complete step
   wzSurvey.MoveTo(step5)
     End If

     Case Else
  'nothing required
    End Select
  End Sub 'wzSurvey_NextButtonClick

   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the wizard's prev button
  ''' click event. It is responsible for altering the survey navigation as
  ''' as function of the answers provided
  ''' </summary>
  '''
  ''' <param name="sender"></param>
  ''' <param name="e"></param>
  Protected Sub wzSurvey_PreviousButtonClick(ByVal sender As Object, _
        ByVal e As WizardNavigationEventArgs)
 'skip steps as a function of the users answers
 Select Case e.CurrentStepIndex
  Case 2
    If ((rbStep1.SelectedIndex >= 0) AndAlso _
        (rbStep3.SelectedItem.Text.Equals("No"))) Then
   'user does not use ASP.NET 2.0 so move to 1.x question
   wzSurvey.MoveTo(step1)
    End If
  Case Else
    'nothing required
 End Select
  End Sub 'wzSurvey_PreviousButtonClick

 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the wizard's active step
 ''' changed event. It is responsible for determining if the survey is
 ''' complete and storing the data in the data store.
 ''' </summary>
 '''
 ''' <param name="sender"></param>
 ''' <param name="e"></param>
 Protected Sub wzSurvey_ActiveStepChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs)
   'check to see if the complete step is the active step
   If (wzSurvey.ActiveStepIndex = wzSurvey.WizardSteps.Count - 1) Then
     'survey is complete so production application should store the data
  'in an applicable data store
   End If
  End Sub  'wzSurvey_ActiveStepChanged
 End Class 'CH04SurveyDataVB1
End Namespace
```

Example 4-12. Simulating a multipage form (.cs)

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

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH04SurveyDataCS1.aspx
 /// </summary>
 public partial class CH04SurveyDataCS1 : System.Web.UI.Page
 {
```

```
 ///*****************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>

    protected void Page_Load(object sender, EventArgs e)
 {
ArrayList yesNoSelections;
ArrayList experienceSelections;

if (!Page.IsPostBack)
{
 // build the arraylist with the yes/no responses
 yesNoSelections = new ArrayList();
 yesNoSelections.Add(new ListItem("Yes", "1"));
 yesNoSelections.Add(new ListItem("No", "0"));

 // bind the yes/no data to the radio button lists in the questions
 rbStep1.DataSource = yesNoSelections;
 rbStep1.DataTextField = "Text";
 rbStep1.DataValueField = "Value";
 rbStep1.DataBind();

 rbStep3.DataSource = yesNoSelections;
 rbStep3.DataTextField = "Text";
 rbStep3.DataValueField = "Value";
 rbStep3.DataBind();

 // build the arraylist with the experience responses
 experienceSelections = new ArrayList();
 experienceSelections.Add(new ListItem("-- Select One --", "0"));
 experienceSelections.Add(new ListItem("0-6 Months", "1"));
 experienceSelections.Add(new ListItem("7-12 Months", "2"));
 experienceSelections.Add(new ListItem("13-24 Months", "3"));
 experienceSelections.Add(new ListItem("24+ Months", "4"));

 // bind the experience data to the radio button lists in the questions
 ddStep2.DataSource = experienceSelections;
 ddStep2.DataTextField = "Text";
 ddStep2.DataValueField = "Value";
 ddStep2.DataBind();

 ddStep4.DataSource = experienceSelections;
 ddStep4.DataTextField = "Text";
 ddStep4.DataValueField = "Value";
 ddStep4.DataBind();
 }
 } // Page_Load
```

```
    ///*********************************************************************
   /// <summary>
  /// This routine provides the event handler for the wizard's next button
  /// click event. It is responsible for altering the survey navigation as
  /// as function of the answers provided
  /// </summary>
  ///
  /// <param name="sender"></param>
  /// <param name="e"></param>
  protected void wzSurvey_NextButtonClick(Object sender,
        WizardNavigationEventArgs e)
  {
  // skip steps as a function of the users answers
  switch (e.CurrentStepIndex)
  {
   case 0:
      if ((rbStep1.SelectedIndex >= 0) &&
          (rbStep1.SelectedItem.Text.Equals("No")))
         {
         // user does not use ASP.NET 1.x so move to 2.0 question
      wzSurvey.MoveTo(step3);
          }
    break;

   case 2:
     if ((rbStep1.SelectedIndex >= 0) &&
              (rbStep3.SelectedItem.Text.Equals("No")))
      {
     // user does not use ASP.NET 2.0 so move to complete step
     wzSurvey.MoveTo(step5);
       }
    break;

   default:
      // nothing required
      break;
  }
 } // wzSurvey_NextButtonClick

   ///*********************************************************************
   /// <summary>
   /// This routine provides the event handler for the wizard's prev button
   /// click event. It is responsible for altering the survey navigation as
   /// as function of the answers provided
   /// </summary>
   ///
   /// <param name="sender"></param>
   /// <param name="e"></param>
   protected void wzSurvey_PreviousButtonClick(Object sender,
           WizardNavigationEventArgs e)
   {
   // skip steps as a function of the users answers
   switch (e.CurrentStepIndex)
   {
   case 2:
      if ((rbStep1.SelectedIndex >= 0) &&
       (rbStep3.SelectedItem.Text.Equals("No")))
   {
      // user does not use ASP.NET 2.0 so move to 1.x question
      wzSurvey.MoveTo(step1);
   }
   break;

  default:
    // nothing required
    break;
  }
   } // wzSurvey_PreviousButtonClick

  ///*******************************************************************
  /// <summary>
  /// This routine provides the event handler for the wizard's active step
  /// changed event. It is responsible for determining if the survey is
  /// complete and storing the data in the data store.
  /// </summary>
  ///
  /// <param name="sender"></param>
  /// <param name="e"></param>


  protected void wzSurvey_ActiveStepChanged(Object sender,
        System.EventArgs e)
  {
    // check to see if the complete step is the active step
 if (wzSurvey.ActiveStepIndex == wzSurvey.WizardSteps.Count - 1)
 {
      // survey is complete so production application should store the data
      // in an applicable data store
 }
  } // wzSurvey_ActiveStepChanged
 } //  CH04SurveyDataCS1
}
```

Recipe 4.5. Setting the Initial Focus to a Specific Control Problem

Problem

You need to set the focus of a page to a specific control when the page is first loaded.

Solution

The simplest solution is to use the `SetFocus` method of the page object to set the focus to a specific control, as shown below:

```
Page.SetFocus(txtFirstName)

Page.SetFocus(txtFirstName);
```

Another more flexible solution is to create a client-side JavaScript block in the code-behind that sets the focus to the desired control and then writes the block to the rendered page so it is executed whe the page is loaded. We focus on this solution from here on because using client-side JavaScript is often helpful, and occasionally a requirement, when working withforms.

In the code-behind class for the page, use the .NET language of your choice to:

1. Write some code that is called from the `Page_Load` method and generates a client-side JavaScript block that calls the `focus` method of the desired control and sets the control's initial focus to itself.
2. Use the `RegisterStartupScript` method of the `ClientScript` object to register the script block so it is output when the page is rendered.

Examples 4-13, 4-14 through 4-15 showthe *.aspx* and code-behind files for the application that implements this solution. (See the "Building a JavaScript Library" sidebar for the rationale behind ou approach.)

Discussion

To implement the JavaScript-based solution, nothing special is required in the *.aspx* file. But the code-behind page is another matter. There, you need to generate a client-side JavaScript block that calls the `focus` method of the desired control and sets the control's initial focus to itself.

The application that we've written to implement the solution uses a simpleform with only three text boxes to capture the user's first name, last name, and age. The application's code-behind assembles a client-side JavaScript block that calls the set focus method of the first text box control and then writes the script block to the rendered page. Here is the code that sets the focus of the first text box on the form, `txtFirstName`:

```
<script type="text/javascript">
<!--
document.getElementById('ctl00_PageBody_txtFirstName').focus();
// -->
</script>
```

The client-side JavaScript block is generated by the `setFocusToControl` method of the code-behind. You pass to `setFocusToControl` the reference to the control that you want to have the focus when the page is initially displayed in the browser, which is done via `controlToFocus.ClientID` in our example. The client-side JavaScript uses the `getElementById` method of the `document` object to get a reference to the passed control and then calls the focus method on the control:

```
scriptBlock = "document.getElementById('" & _
      controlToFocus.ClientID & "').focus();"

scriptBlock = "document.getElementById('" +
      controlToFocus.ClientID + "').focus();";
```

The `RegisterStartupScript` method of the `Page` object is used to register the client-side script block to be output when the page is displayed in the browser. This method outputs the script block at the bottom of the form. This is important because the script block created in the `setFocusToControl` method is executed when the browser parses the page; for it to work correctly, thecontrols on the form have to have been previously created. If this block were output at the top of the page or at the beginning of the form, a JavaScript error would occur because the control to set the focus to would not exist.

To set the initial focus, the `Page_Load` method calls the `setFocusToControl` method when the page is initially loaded, passing it a reference to the control that is to have initial focus.

With the basic functionality in place to set the focus to a control programmatically, many options are available. Refer to Recipe 4.5 for an example that uses the same functionality to set the focus to a control that has a validation error.

Building a JavaScript Library

Client-side JavaScript is frequently required in web pages, and to want to reuse the same JavaScript block is common. In classic ASP, reuse is achieved by using include files or by linking to specific JavaScript files. A big drawback to either of these techniques is that when the files are used as libraries, as is commonly done, they typically contain more functions than needed by any given pages. This results in slowerperformance because the excess code has to be downloaded to the browser along with the required code. Other options include using many files, each containing fewer methods, or putting the functionality required for a specific page directly in that page. Both result in duplication of JavaScript in many places and a maintenance headache when changes are required.

With ASP.NET, you can generate the JavaScript you need by writing specialized methods that you encapsulate in a custom helper class. You call only the methods required to create the JavaScript you need for a specific page. This approach lets you reuse debugged JavaScript for all of your applications, and it improves performance because only the needed functions are rendered in the HTML page. An example of a client script library class that contains the `setFocusToControl` method used in this recipe is shown here. To make the method more useful, we have modified the code to allow you to pass at runtime the identity of the control to receive focus. In addition, the method has the `Shared` attribute to allow calling the method without instantiating the class.

```
 Namespace DDIG.Script
   Public Class ClientScripts
   Private Shared Sub setFocusToControl( _
  ByVal pageControl As System.Web.UI.Page, _
  ByVal scriptManager As System.Web.UI.ClientScriptManager,
  ByVal controlToFocus As System.Web.UI.Control)

   Dim scriptBlock As String

   'add the client script to set the control focus
   scriptBlock = "document.getElementById('" & _
controlToFocus.ClientID & "').focus();"
   'register the client script to be output when the
   'page is rendered
   scriptManager.RegisterStartupScript(pageControl.GetType(), _
            "SetFocusScript", _
         scriptBlock, _
         True)
   End Sub 'setFocusToControl
    End Class
  End Namespace
```

See Also

Recipe 4.5

Example 4-13. Setting focus initially (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH04SetFocusVB2.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH04SetFocusVB2"
  Title="Setting Control Focus - Approach 2" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
   Setting Control Focus - Approach 2 (VB)
 </div>
 <table width="75%" align="center" border="0" >
   <tr>
      <td class="labelText">First Name: </td>
      <td>
         <asp:TextBox id="txtFirstName" Runat="server"
       Columns="30" CssClass="labelText" />
         <asp:RequiredFieldValidator id="rfvFirstName"
      Runat="server"
      ControlToValidate="txtFirstName"
      CssClass="alertText"
      Display="Dynamic"
      EnableClientScript="True">
     <img src="images/arrow_alert.gif"
      alt="arrow"/> First Name Is Required
   </asp:RequiredFieldValidator>
  </td>
   </tr>
 <tr>
    <td class="labelText">Last Name: </td>
       <td>
       <asp:TextBox id="txtLastName" Runat="server"
       Columns="30" CssClass="labelText" />
    <asp:RequiredFieldValidator id="rfvLastName"
      Runat="server"
      ControlToValidate="txtLastName"
      CssClass="alertText"
      Display="Dynamic"
      EnableClientScript="True">
        <img src="images/arrow_alert.gif"
      alt="arrow"/> Last Name Is Required
   </asp:RequiredFieldValidator>
  </td>
   </tr>
  <tr>
  <td class="labelText">Age: </td>
  <td>
    <asp:TextBox id="txtAge" Runat="server"
     Columns="30" CssClass="labelText" />
          <asp:RequiredFieldValidator id="rfvAge"
     Runat="server"
     ControlToValidate="txtAge"
     CssClass="alertText"
     Display="Dynamic"
     EnableClientScript="True">
      <img src="images/arrow_alert.gif"
     alt="arrow"/> Age Is Required
   </asp:RequiredFieldValidator>
   <asp:RangeValidator id="rvAge" Runat="server"
       ControlToValidate="txtAge"
       CssClass="alertText"
       Display="Dynamic"
       EnableClientScript="True"
       MinimumValue="18"
       MaximumValue="99"
       Type="Integer">
  <img src="images/arrow_alert.gif"
    alt="arrow"/> Age Must Be Between 18 and 99
   </asp:RangeValidator>
  </td>
   </tr>
  <tr>
  <td colspan="2">
      <br/>
   <table align="center" width="50%">
    <tr>
      <td align="center">
       <input id="btnSave" runat="server" type="button"
       value="Save" causesvalidation="true"
       onserverclick="btnSave_ServerClick"/>
     </td>
    <td align="center">
       <input id="btnCancel" runat="server" type="button"
     value="Cancel" causesvalidation="false"/>
    </td>
    </tr>
    </table>
    </td>
  </tr>
   </table>
</asp:Content>
```

Example 4-14. Setting focus initially (.vb)

```
Option Explicit
On Option Strict On

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
   '''  CH04SetFocusVB2.aspx
  ''' </summary>
Partial  Class  CH04SetFocusVB2
 Inherits System.Web.UI.Page

 '''*****************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
   If (Not Page.IsPostBack) Then
  setFocusToControl(txtFirstName)
   End If
    End Sub 'Page_Load

 '''*****************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the save button click
 ''' event.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnSave_ServerClick(ByVal sender As Object, _
      ByVal e As System.EventArgs)
   If (Page.IsValid) Then
  'process form data and save as required for application
   End If
 End Sub 'btnSave_ServerClick

   '''*****************************************************************************
   ''' <summary>
   ''' This routine generates the client script to set the focus to the
   ''' passed control
   ''' </summary>
   '''
   ''' <param name="controlToFocus">Set to the control to which focus is
   '''        to be set</param>
   Private Sub setFocusToControl(ByVal controlToFocus As System.Web.UI.Control)
  Dim scriptBlock As String

  'add the client script to set the control focus
    scriptBlock = "document.getElementById('" & _
    controlToFocus.ClientID & "').focus();"

   'register the client script to be output when the page is rendered
   ClientScript.RegisterStartupScript(Me.GetType(), _
       "SetFocusScript", _
        scriptBlock, _
        True)

   End Sub 'setFocusToControl
     End Class 'CH04SetFocusVB2
 End Namespace
```

Example 4-15. Setting focus initially (.cs)

```
using System;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH04SetFocusCS2.aspx
 /// </summary>
 public partial class CH04SetFocusCS2 : System.Web.UI.Page
 {

  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
  if (!Page.IsPostBack)
  {
   setFocusToControl(txtFirstName);
  }
 } // Page_Load

    ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the save button click
```

```
  /// event.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnSave_ServerClick(Object sender,
          System.EventArgs e)
  {
   if (Page.IsValid)
   {
      // process form data and save as required for application
   }
  } //btnSave_ServerClick

  ///*******************************************************************
  /// <summary>
  /// This routine generates the client script to set the focus to the
  /// passed control
  /// </summary>
  ///
  /// <param name="controlToFocus">Set to the control to which focus is
  /// to be set</param>
  private void setFocusToControl(System.Web.UI.Control controlToFocus)
  {
   String scriptBlock;

   // add the client script to set the control focus
   scriptBlock = "document.getElementById('" +
      controlToFocus.ClientID + "').focus();";

   // register the client script to be output when the page is rendered
   ClientScript.RegisterStartupScript(this.GetType(),
          "SetFocusScript",
          scriptBlock,
          true);

  } // setFocusToControl
   } // CH04SetFocusCS2
 }
```

PREV

Recipe 4.6. Setting the Focus to a Control with a Validation Error

Problem

You want to set the focus to the first control on your form that has a validation error.

Solution

The solution to this recipe is an extension of the one introduced in Recipe 4.4, where we recommend writing code that generates some client-side JavaScript that calls the focus method of the desired control. For this recipe, we recommend adding some additional JavaScript-generating code tied to th Save button's click event handler and, when executed, searches for a control with a validation error and sets the focus to that control.

In the code-behind class for the page, use the .NET language of your choice to:

1. Write some code that is called from the `Page_Load` method and generates a client-side JavaScript block that calls the focus method of the desired control and sets the control's initial focus to itself (see Recipe 4.4 for details).
2. Add additional code to the Save (or equivalently named) button's click event handler that searches for a control with a validation error and sets the focus to that control.
3. Use the `RegisterStartupScript` method of the `ClientScript` object to register the script block so that it is output when the page is rendered.

Examples 4-16 and 4-17 show routines required to implement the last two steps of this solution.

Discussion

As described in Recipe 4.4, you start implementing this solution by creating a client-side JavaScript block in the code-behind that sets the focus to a desired control and then outputs the block to the rendered page so it is executed when the page is loaded. With this code in place, you can add some additional code to the Save (or equivalently named) button's click event that determines the first control with a validation error and sets the focus to it via the previously loaded JavaScript code.

This solution relies on the page object containing a collection of the validation controls on the form. The order of the validators in the collection is the same as the order in which they appear in the *.aspx* file. (The validators should be placed with the controls they validate for this solution to work correctly.)

To get a feel for how to implement this solution, first take a look at the sample application we create for Recipe 4.4 (Examples 4-13, 4-14 through 4-15). Next, consider the code in Examples 4-16 and 4-17, which is added to that application to implement this recipe's solution.

With the additional code, when the server-side button's click event is executed, a check is first made to see if the page is valid. If it is, a save operation will be performed. If the page is invalid, the validators will be iterated through until the first invalid one is found. This is determined by examining the `IsValid` property of each validator control; its value will be `false` if the control associated with the validator has failed to pass validation.

When an invalid validator is found, the associated control is identified by calling the `FindControl` method of the `validator` object. The control is then passed to the `setFocusToControl` method. Only one control can have the focus, so after an invalid control is found, the `for` loop is exited.

The client-side validation is disabled in this example to simplify the explanation of how to determine which control has a validation error and how to set the focus to it. If we had kept client-side validation enabled, we would have had to implement the same approach using client-side JavaScript. Though the latter may prove to be the most useful for you, we have avoided it here to keep this recipe lean and to the point.

See Also

Recipe 4.4

Example 4-16. Setting focus to control with validation error (.vb)

```
'''*************************************************************************
''' <summary>
''' This routine provides the event handler for the save button click
''' event.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnSave_ServerClick(ByVal sender As Object, _
      ByVal e As System.EventArgs)
  Dim validator As System.Web.UI.WebControls.BaseValidator

  If (Page.IsValid) Then
 'process form data and save as required for application

  Else
 'page is invalid so iterate through validators to find the first one
 'with an error
 For Each validator In Page.Validators
    If (Not validator.IsValid) Then
```

```
      'validator that failed found so set the focus to the control
   'it validates and exit the loop
   setFocusToControl(validator.FindControl(validator.ControlToValidate))
   Exit For
   End If
 Next validator
  End If
End Sub 'btnSave_ServerClick
```

Example 4-17. Setting focus to control with validation error (.cs)

```
///***************************************************************************
/// <summary>
/// This routine provides the event handler for the save button click
/// event.
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void btnSave_ServerClick(Object sender,
       System.EventArgs e)
{
   if (Page.IsValid)
   {
  // process form data and save as required for application
   }
   else
   {
  // page is invalid so iterate through validators to find the first one
  // with an error
  foreach (BaseValidator validator in Page.Validators)
  {
  if (!validator.IsValid)
  {
  // validator that failed found so set the focus to the control
  // it validates and exit the loop
  setFocusToControl(validator.FindControl(validator.ControlToValidate));
  break;
   }
     }
   }
} //btnSave_ServerClick
```

PREV

Chapter 5. User Controls

PREV

PREV

5.0 Introduction

User controls provide an excellent mechanism for code reuse in ASP.NET. Indeed, the reuse mechanism is better than the server-side include file method used in classic ASP. For one thing, user controls are compiled and can be cached separately from thepage in which they are used, providing an increase in performance. (See Chapter 19.) For another, user controls leverage the object model support provided by ASP.NET, which means that you can program against any properties you declare for the control, just like other ASP.NET server controls.

This brings us to another point: user controls exist only on the server. When rendered and sent to the client, they are just part of the flat HTML for a page. Since they exist only on the server, browser compatibility is not an issue. It is unnecessary to download a control from the server to the client and, in the process, risk that a user might refuse to download a control needed to render your web page properly.

PREV

PREV

Recipe 5.2. Sharing a Page Header on Multiple Pages

Problem

You have a header common to all pages in a site and do not want to repeat the code on every page.

Solution

Create a user control containing the header code and add the control to each page.

To create the user control:

1. Create a file with a *.ascx* extension.
2. Place the `@ Control` directive at the top of the file.
3. Add the HTML you wish to reuse for the header, making sure to avoid using any `<html>`, `<head>`, `<body>`, or `<form>` elements.

In the code-behind class for the control, use the .NET language of your choice to:

1. Create a user control code-behind class that inherits from the `UserControl` class.
2. (Optional) Establish properties for the control that will provide the ability to control the basic look of the header programmatically.

To use the user control in an ASP.NET page:

1. Register the control by using the `@ Register` directive at the top of the page.
2. Place the tag for the user control where you want the control to be rendered.

To demonstrate this solution, we've created a user control that houses some typical header HTML, including `<img>` tags for a header image and a divider image. We then show how to use the user control in three different ways: with default parameters, with parameters set in the *.aspx* file, and with parameters set in the code-behind.

The output of a test page that uses the user control is shown in Figure 5-1. (The second divider line appears in green and the third divider line appears in blue when rendered on the screen.) Example 5-

1 shows the *.ascx* file for the user control. Examples 5-2 and 5-3 show the VB and C# code-behind files for the user control. Example 5-4 shows the *.aspx* file that uses the user control in the three different ways previously mentioned. Examples 5-5 and 5-6 show the VB and C# code-behind files for the test page that uses the user control.

Discussion

User controls provide an easy way to partition pages and reuse the page sections throughout your application. They are similar to web forms in that they consist of two files: a *.ascx* file (the user interface) and a code-behind file (*.vb* or *.cs*). Since they constitute page sections, the user interface elements they contain generally do not include `<html>, <head>, <body>`, or `<form>` elements. Further, you must place the `@ Control` directive at the top of the *.ascx* file instead of the `@ Page` directive used for web forms. The code-behind class for a user control is similar to the code-behind class for a web form, with the most noticeable difference being that the user control code-behind class inherits from the `UserControl` class instead of the `Page` class.

The example we've provided to demonstrate this solution shows you how to create a user control to be used as a page header. The user control has three custom properties that provide the ability to control the basic look of the header programmatically.

The HTML for the user control (see Example 5-1) contains a table with two rows and a single column. The first row of the table contains a "logo" style image. The `src` attribute of the image tag is set to the name of the image that will be used for the default image.

Figure 5-1. Page header user control output

The cell in the second row of the table contains a single-pixel transparent image. This is a classic

HTML trick to allow manipulation of a table cell to provide dividers with alterable colors and heights a well as to stretch the size of the cell. The `bgcolor` and `height` attributes of the cell are set to the values that will be the defaults for the color and height of the divider line.

The code-behind contains three properties:

`headerImage`

Provides the ability to set/get the image that will be displayed in the header

`dividerColor`

Provides the ability to set/get the divider color

`dividerHeight`

Provides the ability to set/get the divider height

To use a user control in an ASP.NET page, the control must be registered using the `@ Register` directive at the top of the page. The `TagPrefix` and `TagName` attributes are used to define the "custom tag" uniquely on the ASP.NET page. The `TagPrefix` is typically set to the project's namespace. The `TagName` should be set to a value that describes the control's use. The `Src` attribute is set to the name of the *.ascx* file of the user control. Here is how we set these attributes in our example:

```
<%@ Register TagPrefix="ASPCookbook" TagName="PageHeader"
Src="CH05UserControlHeaderVB.ascx"  %>
```

Additionally, a tag is placed in the HTML where you want the user control rendered. The tag must be given an `id` and the `runat` attribute must be set to `"server"` or else the user control will not be rendered. Here is the syntax we've used for inserting our example user control using the default values:

```
<ASPCookbook:PageHeader  id="pageHeader1"  runat="server"  />
```

If you want to change the properties for a user control, you can set them as attributes in the tag. Fo instance, here is how we set the header image, the divider color, and the divider height in our example:

```
<ASPCookbook:PageHeader  id="pageHeader2"  runat="server"
      headerImage="images/oreilly_header.gif"
       dividerColor="#008000"
      dividerHeight="12" />
```

The properties for a user control included in an ASP.NET page can be set in the code-behind in the same manner as setting the attributes for any other HTML or ASP.NET server control. You can set the appropriate values in `Page_Load` or another method. Here's how we do it in our example:

VB

```
 pageHeader3.headerImage  =  "images/ddig_logo.gif"
 pageHeader3.dividerHeight  =  "18"
 pageHeader3.dividerColor  =  ColorTranslator.ToHtml(Color.DarkBlue)
```

C#

```
 pageHeader3.headerImage  =  "images/ddig_logo.gif";
 pageHeader3.dividerHeight  =  "18";
 pageHeader3.dividerColor  =  ColorTranslator.ToHtml(Color.DarkBlue);
```

See Also

Recipe 1.1 for how to use master pages to share common HTML between pages

Example 5-1. Page header user control (.ascx)

```
<%@ Control Language="VB" AutoEventWireup="false"
 CodeFile="CH05UserControlHeaderVB.ascx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH05UserControlHeaderVB"   %>
<table  width="100%"  cellpadding="0"  cellspacing="0"  border="0">
 <tr>
  <td align="center">
   <img id="imgHeader" runat="server"
    src="images/ASPNetCookbookHeading_blue.gif"
    alt="ASPNETCookbook"/>
  </td>
 </tr>
 <tr>
  <td  id="tdDivider"  runat="server"  bgcolor="#6B0808"  height="6">
   <img src="images/spacer.gif" alt="divider"/></td>
 </tr>
</table>
```

Example 5-2. Page header user control (.vb)

```
Option Explicit On
Option Strict On

Imports System.Web.UI.WebControls

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH05UserControlHeaderVB.ascx
 ''' </summary>
 Partial Class CH05UserControlHeaderVB
  Inherits System.Web.UI.UserControl

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability get/set the image used in the
  ''' header user control
  ''' </summary>
  Public Property headerImage() As String
   Get
    Return (imgHeader.Src)
   End Get

   Set(ByVal Value As String)
    imgHeader.Src = Value
   End Set
  End Property 'headerImage

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability get/set the divider color used
  ''' at the bottom of the user control
  ''' </summary>
  Public Property dividerColor() As String
   Get
    Return (tdDivider.BgColor)
   End Get

   Set(ByVal Value As String)
    tdDivider.BgColor = Value
   End Set
  End Property 'dividerColor

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability get/set the divider height used
  ''' at the bottom of the user control
  ''' </summary>
  Public Property dividerHeight() As String
   Get
    Return (tdDivider.Height)
   End Get
```

```
    Set(ByVal Value As String)
     tdDivider.Height = Value
    End Set
   End Property 'dividerHeight

   '''*************************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the page load event. It
   ''' is responsible for initializing the controls on the page.
   ''' </summary>
   '''
   ''' <param name="sender">Set to the sender of the event</param>
   ''' <param name="e">Set to the event arguments</param>
   Protected Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
    'place user code here
   End Sub 'Page_Load
  End Class 'CH05UserControlHeaderVB
End Namespace
```

Example 5-3. Page header user control (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH05UserControlHeaderCS.ascx
 /// </summary>
 public partial class CH05UserControlHeaderCS : System.Web.UI.UserControl
 {
   ///************************************************************************
   /// <summary>
   /// This property provides the ability get/set the image used in the
   /// header user control
   /// </summary>
   public String headerImage
   {
    get
    {
     return imgHeader.Src;
    }

    set
    {
     imgHeader.Src = value;
    }
   } // headerImage

   ///***************************************************************************
   /// <summary>
   /// This property provides the ability get/set the divider color used
   /// at the bottom of the user control
   /// </summary>
   public String dividerColor
   {
    get
    {
     return tdDivider.BgColor;
    }

    set
    {
     tdDivider.BgColor = value;
    }
   } // dividerColor

   ///***************************************************************************
   /// <summary>
   /// This property provides the ability get/set the divider height used
   /// at the bottom of the user control
   /// </summary>
   public String dividerHeight
   {
    get
    {
     return tdDivider.Height;
    }

    set
    {
     tdDivider.Height = value;
    }
   } // dividerHeight

   ///***************************************************************************
   /// <summary>
   /// This routine provides the event handler for the page load event.
   /// It is responsible for initializing the controls on the page.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void Page_Load(object sender, EventArgs e)
   {
    // place user code here
   } // Page_Load
  } //  CH05UserControlHeaderCS
```

```
}
```

Example 5-4. Using the page header user control (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH05DisplayHeaderVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH05DisplayHeaderVB"
 Title="User Control Display Header" %>
<%@ Register TagPrefix="ASPCookbook" TagName="PageHeader"
       Src="CH05UserControlHeaderVB.ascx" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <br />
 <br />
 <table width="90%" align="center" border="0">
  <tr>
   <td class="pageHeading">Header Using Default Parameters:</td>
  </tr>
  <tr>
   <td>
    <ASPCookbook:PageHeader id="pageHeader1" runat="server" />
   </td>
  </tr>
  <tr>
   <td class="pageHeading"><br /><br />
    Header With Parameters Set In ASPX:</td>
  </tr>
  <tr>
   <td>
    <ASPCookbook:PageHeader id="pageHeader2" runat="server"
         headerImage="images/oreilly_header.gif"
         dividerColor="#008000"
         dividerHeight="12" />
   </td>
  </tr>
  <tr>
   <td class="pageHeading">
    <br /><br />
    Header With Parameters Set In Code-behind:</td>
  </tr>
  <tr>
   <td>
    <ASPCookbook:PageHeader id="pageHeader3" runat="server" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 5-5. Using the page header user control (.vb)

```
Option Explicit On
Option Strict On

Imports System.Drawing

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
  '''  CH05DisplayHeaderVB.aspx
  ''' </summary>
  Partial Class CH05DisplayHeaderVB
    Inherits System.Web.UI.Page

     '''*********************************************************************
    ''' <summary>
    ''' This routine provides the event handler for the page load event. It
    ''' is responsible for initializing the controls on the page.
    ''' </summary>
    '''
    ''' <param name="sender">Set to the sender of the event</param>
    ''' <param name="e">Set to the event arguments</param>
    Protected Sub Page_Load(ByVal sender As Object, _
       ByVal e As System.EventArgs) Handles Me.Load
   'initialize the 3rd page header user control
   pageHeader3.headerImage = "images/ddig_logo.gif"
   pageHeader3.dividerHeight = "18"
   pageHeader3.dividerColor = ColorTranslator.ToHtml(Color.DarkBlue)
        End Sub 'Page_Load
 End Class 'CH05DisplayHeaderVB
End Namespace
```

Example 5-6. Using the page header user control (.cs)

```
using System;
using System.Drawing;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH05DisplayHeaderCS.aspx
 /// </summary>
 public partial class CH05DisplayHeaderCS : System.Web.UI.Page
 {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   // initialize the 3rd page header user control
   pageHeader3.headerImage = "images/ddig_logo.gif";
   pageHeader3.dividerHeight = "18";
   pageHeader3.dividerColor = ColorTranslator.ToHtml(Color.DarkBlue);
  } // Page_Load
 } // CH05DisplayHeaderCS
}
```

Recipe 5.3. Creating a Customizable Navigation Bar

Problem

You want to create a navigation bar that lets you add or remove items without changing code so you can reuse the navigation bar in multiple applications.

Solution

Create an XML document containing the items that will be displayed in the navigation bar, and then create a user control that uses the contents of the XML document to provide the required customization.

To create the user control:

1. Create a file with a *.ascx* extension.
2. Place the `@ Control` directive at the top of the file.
3. Add a `DataList` control configured to render a table with an `ItemTemplate` defining the cells in the table.

In the code-behind class for the control, use the .NET language of your choice to:

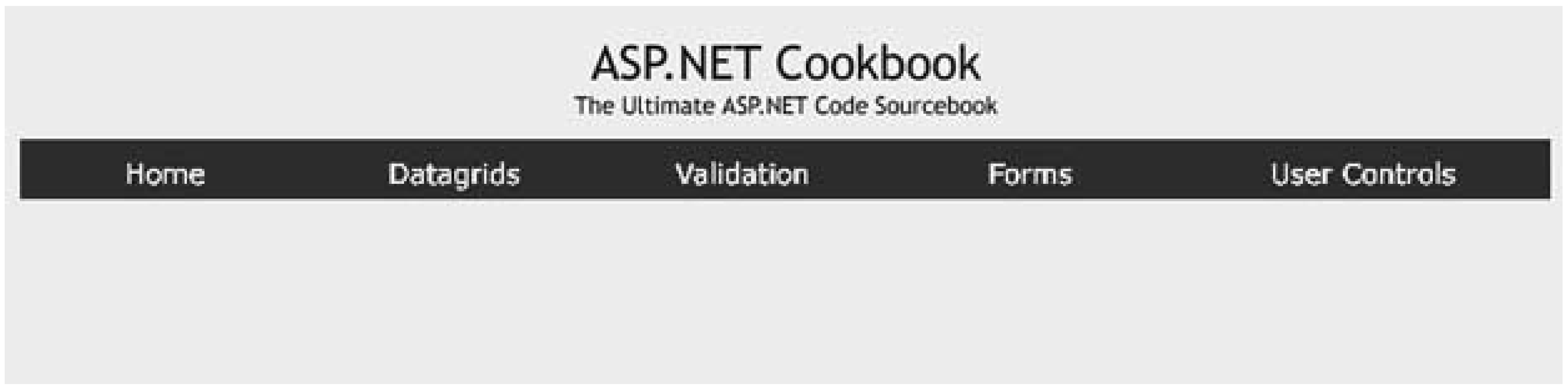
1. Create a user control code-behind class that inherits from the `UserControl` class.
2. (Optional) Establish properties for the control that will provide the ability to control programmatically the basic look of the navigation bar, such as its background color.

To use the user control in an ASP.NET page:

1. Register the control by using the `@ Register` directive at the top of the page.
2. Place the tag for the user control in the HTML where you want the control rendered.

The output of a test page demonstrating a typical navigation bar user control is shown in Figure 5-2. Example 5-7 shows the XML document we created to define the contents of the navigation bar. Example 5-8 shows the *.ascx* file for the user control. Examples 5-9 and 5-10 show the VB and C# code-behind files for the user control. Example 5-11 shows the *.aspx* file for the test page that uses the user control. Examples 5-12 and 5-13 show the VB and C# code-behind files for the test page.

Figure 5-2. Customizable navigation bar output



Discussion

The general strategy for this solution is to create an XML document that defines the contents of the navigation bar user control. You then use the `Page_Load` method of the user control's code-behind to read into a `DataSet` the XML document containing the navigation bar data and then bind the dataset to a `DataList` control. Using a `DataSet` and `DataList` control in this way has three advantages:

1. You are not limited to the number of items in the navigation bar.
2. Loading the XML document used to define the navigation bar items into a`DataSet` makes for easy traversal of those items.
3. A `DataList` control configured to render a table with an `ItemTemplate` provides the flexibility in the display of the columns that is needed for customizing a navigation bar.

The example we have written to implement this solution creates a navigation bar user control whose contents are defined by an XML document. But the example goes a bit further in that it provides the ability to define what buttons appear in the navigation bar as well as the ability to customize the colo of the bar, change the name of the XML file used to define the bar, and set other properties.

The XML document that defines our navigation bar consists of a series of elements named`Public`, as shown in Example 5-7. This is the name of the navigation bar (explained later). Each of the`Public` elements contains three elements: the `ButtonLink` element defines the URL for the navigation bar button, the `ImageSrc` element defines the image used for the button, and the `AltText` element sets the text alternative for the image (i.e., the value of the`Alt` attribute of the `IMG` tag used for the button).

The *.ascx* file for our example user control, which is shown inExample 5-8, contains a `DataList` control configured to render a table with an `ItemTemplate` to define the cells in the table. The `ItemTemplate` contains an anchor tag used for the navigation and an image tag to display the graphic button.

The code-behind for our example user control, shown inExamples 5-9 (VB) and 5-10 (C#), contains three properties to enable customization of the navigation bar. The`backgroundColor` property provides the ability to change the background color of the navigation bar. The`xmlFilename` property defines the XML document used to populate the navigation bar. The `navBarName` property is used to define the name of the group of elements in the XML document used to populate the navigation bar. In this example, all of the elements are named`Public`. The example's design allows the XML document to have any number of other element groups, thus providing the ability to have a different navigation bar on different pages depending on the page type,user role, context, or the like. If you wanted a different navigation bar for the private pages in the site, for example, you would add a group of `Private` elements with the information needed to define the private navigation bar. How to select one is described later.

Our approach advocates leveraging a `DataList` tabular control for the workings of the navigation bar. Here's how we populate the `DataList`:

1. In the `Page_Load` method of the code-behind, the XML document containing the navigation bar data is read into a `DataSet` and then bound to the `DataList`.

2. The `dlNavBar_ItemDataBound` method is then called by ASP.NET for each of the items defined in the XML document (rows in the `DataSet`). Its job is to set the `HRef` of the anchor tag and then set the image source and `alt` text for the image tag.

To use the navigation bar user control in an *.aspx* page, the control must be registered with the `@ Register` directive at the top of the page and then the navigation bar control can be inserted into your page. Example 5-11 shows how this is done in our application, including the use of`TagPrefix`, `TagName`, and `Src` attributes set to the namespace of the project, the name of the control, and the name (and virtual path) of the *.ascx* file of the user control, respectively (see Recipe 5.1 for more details on these attributes).

In our example, the three properties of the navigation bar control must be set. Setting these in the *.aspx* file is possible; however, because the `xmlFilename` property must be set to a fully qualified XML filename, this is better done in the code-behind as shown inExamples 5-12 (VB) and 5-13 (C#).

The navigation bar user control presented here is somewhat bland compared to most others. For instance, many navigation bars we have implemented support a changing image to indicate the active location in the site, complete with mouse-overs for each new image. When implementing this capability yourself, consider adding additional image information in the XML document to support the "on," "off," and "over" images. The typical mouse-over code will need to be added to the*.ascx* file, and the code-behind will need a`currentPage` property to provide the ability for it to change the images displayed as a function of the currently displayed page.

The performance of the navigation control shown in this recipe can be improved by caching the control, as described in Recipe 16.6.

See Also

Recipe 16.6 for caching user controls

Example 5-7. XML used for customizable navigation bar

```
<?xml version="1.0" encoding="utf-8"?>
<NavBar>
  <Public>
    <ButtonLink>../ChapterMenu.aspx</ButtonLink>
 <ImageSrc>images/nav/button_nav_home_off.gif</ImageSrc>
 <AltText>Home</AltText>
  </Public>
  <Public>
     <ButtonLink>../ProblemMenu.aspx?Chapter=2</ButtonLink>
 <ImageSrc>images/nav/button_nav_datagrids_off.gif</ImageSrc>
 <AltText>Datagrids</AltText>
  </Public>
  <Public>
     <ButtonLink>../ProblemMenu.aspx?Chapter=3</ButtonLink>
 <ImageSrc>images/nav/button_nav_validation_off.gif</ImageSrc>
 <AltText>Validation</AltText>
  </Public>
  <Public>
     <ButtonLink>../ProblemMenu.aspx?Chapter=4</ButtonLink>
 <ImageSrc>images/nav/button_nav_forms_off.gif</ImageSrc>
 <AltText>Forms</AltText>
  </Public>
  <Public>
     <ButtonLink>../ProblemMenu.aspx?Chapter=5</ButtonLink>
 <ImageSrc>images/nav/button_nav_user_controls_off.gif</ImageSrc>
 <AltText>User Controls</AltText>
  </Public>
</NavBar>
```

Example 5-8. Customizable navigation bar (.ascx)

```
<%@ Control Language="VB" AutoEventWireup="false"
   CodeFile="CH05UserControlNavBarVB.ascx.vb"
   Inherits="ASPNetCookbook.VBExamples.CH05UserControlNavBarVB"  %>
<asp:datalist id="dlNavBar" runat="server"
     width="100%" borderwidth="0"
     cellpadding="0" cellspacing="0" height="29"
     repeatdirection="Horizontal" repeatlayout="Table"
     OnItemDataBound="dlNavBar_ItemDataBound">
  <itemtemplate>
    <td height="25" align="center">
   <a id="anNavBarLink" runat="server" >
   <img id="imgNavBarImage" runat="server" border="0"
     alt="" src=""/></a>
 </td>
  </itemtemplate>
</asp:datalist>
```

Example 5-9. Customizable navigation bar (.vb)

```
Option Explicit On
Option Strict On

Imports System.Data
Imports System.Web.UI.WebControls

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
  ''' CH05UserControlNavBarVB.ascx
  ''' </summary>
   Partial Class CH05UserControlNavBarVB
    Inherits System.Web.UI.UserControl

 'private attributes
 Private mXMLFilename As String
 Private mNavBarName As String

 'The following constants define the elements available
 'in the navigation bar XML document
 Private Const MENU_ITEM_BUTTON_LINK As String = "ButtonLink"
 Private Const MENU_ITEM_IMAGE_SRC As String = "ImageSrc"
 Private Const MENU_ITEM_ALT_TEXT As String = "AltText"

 '''**************************************************************************
 ''' <summary>
 ''' This property provides the ability get/set the background color used
 ''' for the navigation bar
 ''' </summary>
 Public Property backgroundColor() As System.Drawing.Color
   Get
     Return (dlNavBar.BackColor)
   End Get

   Set(ByVal value As System.Drawing.Color)
     dlNavBar.BackColor = value
   End Set
 End Property

 '''************************************************************************
 ''' <summary>
 ''' This property provides the ability get/set the name of the xml file
 ''' used to define the navigation bar
 ''' </summary>
 Public Property xmlFilename() As String
   Get
     Return (mXMLFilename)
   End Get

   Set(ByVal value As String)
     mXMLFilename = value
   End Set
    End Property

 '''************************************************************************
 ''' <summay>
 ''' This property provides the ability get/set the name of the  navigation
 ''' bar definition in the xml file
 ''' </summary>
 Public Property navBarName() As String
     Get
    Return (mNavBarName)
  End Get

  Set(ByVal value As String)
    mNavBarName = value
  End Set
 End Property

 '''************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
    Dim dsNavBarData As DataSet

    'load the XML document used to define the navigation bar items
    'into a dataset to provide easy traversal
    dsNavBarData = New DataSet
    dsNavBarData.ReadXml(xmlFilename)

    'bind the nav bar data to the repeater on the control
    dlNavBar.DataSource = dsNavBarData.Tables(navBarName)
    dlNavBar.DataBind( )
  End Sub 'Page_Load

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the item data bound event
  ''' of the datalist control in the nav bar. It is responsible for setting
  ''' the anchor and image attributes for the item being bound.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub dlNavBar_ItemDataBound(ByVal sender As Object, _
           ByVal e As DataListItemEventArgs)
   'the following constants define the names of the controls in the datalist
   Const ANCHOR_CONTROL As String = "anNavBarLink"
   Const IMAGE_CONTROL As String = "imgNavBarImage"

   Dim anchorControl As HtmlAnchor
   Dim imageControl As HtmlImage
   Dim dRow As DataRowView

   'make sure this is an item or alternating item in the repeater
   If ((e.Item.ItemType = ListItemType.Item) Or _
    (e.Item.ItemType = ListItemType.AlternatingItem)) Then
     'get the data being bound
     dRow = CType(e.Item.DataItem, _
       DataRowView)

     'find the link control then set it to the url
     anchorControl = CType(e.Item.FindControl(ANCHOR_CONTROL), _
        HtmlAnchor)
     anchorControl.HRef = CStr(dRow.Item(MENU_ITEM_BUTTON_LINK))

     'find the image control then set the image source and alt text
     imageControl = CType(e.Item.FindControl(IMAGE_CONTROL), _
        HtmlImage)
     imageControl.Src = CStr(dRow.Item(MENU_ITEM_IMAGE_SRC))
     imageControl.Alt = CStr(dRow.Item(MENU_ITEM_ALT_TEXT))
    End If
   End Sub 'dlNavBar_ItemDataBound
  End Class 'CH05UserControlNavBarVB
End Namespace
```

Example 5-10. Customizable navigation bar (.cs)

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

namespace ASPNetCookbook.CSExamples
  {
    /// <summary>
 /// This class provides the code behind for
 ///  CH05UserControlNavBarCS.ascx
 /// </summary>
 public partial class CH05UserControlNavBarCS : System.Web.UI.UserControl
 {
   // private attributes
   private String mXMLFilename;
   private String mNavBarName;

   // The following constants define the elements available
   // in the navigation bar XML document
   private const String MENU_ITEM_BUTTON_LINK = "ButtonLink";
   private const String MENU_ITEM_IMAGE_SRC = "ImageSrc";
   private const String MENU_ITEM_ALT_TEXT = "AltText";

   ///**************************************************************************
   /// <summary>
   /// This property provides the ability get/set the background color used
   /// for the navigation bar
   /// </summary>
   public System.Drawing.Color backgroundColor
   {
  get
  {
    return dlNavBar.BackColor;
  }

  set
  {
    dlNavBar.BackColor = value;
  }
   } // backgroundColor

   ///**************************************************************************
   /// <summary>
   /// This property provides the ability get/set the name of the xml file
   /// used to define the navigation bar
   /// </summary>
   public String xmlFilename
   {
  get
  {
    return mXMLFilename;
  }

  set
  {
    mXMLFilename = value;
  }
    } // xmlFilename

     ///*************************************************************************
    /// <summary>
    /// This property provides the ability get/set the name of the navigation
    /// bar definition in the xml file
    /// </summary>
    public String navBarName
    {
    get
    {
      return mNavBarName;
    }

    set
    {
      mNavBarName = value;
   }
    } // navBarName

   ///*************************************************************************
   /// <summary>
   /// This routine provides the event handler for the page load event.
   /// It is responsible for initializing the controls on the page.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void Page_Load(object sender, EventArgs e)
   {
      DataSet dsNavBarData;

   // load the XML document used to define the navigation bar items
   // into a dataset to provide easy traversal
   dsNavBarData = new DataSet();
   dsNavBarData.ReadXml(xmlFilename);

   // bind the nav bar data to the repeater on the control
   dlNavBar.DataSource = dsNavBarData.Tables[navBarName];
   dlNavBar.DataBind();
```

```
    } // Page_Load

     ///***************************************************************************
    /// <summary>
    /// This routine provides the event handler for the item data bound event
    /// of the datalist control in the nav bar. It is responsible for setting
    /// the anchor and image attributes for the item being bound.
    /// </summary>
    ///
    /// <param name="sender">Set to the sender of the event</param>
    /// <param name="e">Set to the event arguments</param>
    protected void dlNavBar_ItemDataBound(Object sender,
          DataListItemEventArgs e)
    {
   // the following constants define the names of the controls in the datalist
   const String ANCHOR_CONTROL = "anNavBarLink";
   const String IMAGE_CONTROL = "imgNavBarImage";

   HtmlAnchor anchorControl;
   HtmlImage imageControl;
   DataRowView dRow;

   // make sure this is an item or alternating item in the repeater
   if ((e.Item.ItemType == ListItemType.Item) ||
    (e.Item.ItemType == ListItemType.AlternatingItem))
   {
     // get the data being bound
     dRow = (DataRowView)(e.Item.DataItem);
    // find the link control then set it to the url
    anchorControl = (HtmlAnchor)(e.Item.FindControl(ANCHOR_CONTROL));
    anchorControl.HRef = (String)(dRow[MENU_ITEM_BUTTON_LINK]);

    // find the image control then set the image source and alt text
    imageControl = (HtmlImage)(e.Item.FindControl(IMAGE_CONTROL));
    imageControl.Src = (String)(dRow[MENU_ITEM_IMAGE_SRC]);
       imageControl.Alt = (String)(dRow[MENU_ITEM_ALT_TEXT]);
      }
   } // dlNavBar_ItemDataBound
   } // CH05UserControlNavBarCS
}
```

Example 5-11. Using the navigation bar (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH05DisplayNavBarVB.aspx.vb"
   Inherits="ASPNetCookbook.VBExamples.CH05DisplayNavBarVB"
  Title="User Control Display Navigation Bar" %>
<%@ Register TagPrefix="ASPCookbook" TagName="NavBar"
    Src="CH05UserControlNavBarVB.ascx" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
      <table width="100%" align="center" border="0"
       cellpadding="0" cellspacing="0" >
        <tr>
      <td>
           <ASPCookbook:NavBar id="navBar" runat="server" />
      </td>
    </tr>
  </table>
</asp:Content>
```

Example 5-12. Using the navigation bar (.vb)

```
Option Explicit On
Option Strict On

Imports System.Drawing

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
  ''' CH05DisplayNavBarVB.aspx
  ''' </summary>
  Partial Class CH05DisplayNavBarVB
    Inherits System.Web.UI.Page
 '''*********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
  'initialize the navbar user control
  navBar.xmlFilename = Server.MapPath("xml") & "\NavigationBar.xml"
  navBar.navBarName = "Public"
  navBar.backgroundColor = ColorTranslator.FromHtml("#6B0808")
 End Sub 'Page_Load
  End Class 'CH05DisplayNavBarVB
End Namespace
```

Example 5-13. Using the navigation bar (.cs)

```
using System;
using System.Drawing;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH05DisplayNavBarCS.aspx
 /// </summary>
 public partial class CH05DisplayNavBarCS : System.Web.UI.Page
 {
   ///***********************************************************************
   /// <summary>
   /// This routine provides the event handler for the page load event.
   /// It is responsible for initializing the controls on the page.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void Page_Load(object sender, EventArgs e)
   {
     // initialize the navbar user control
  navBar.xmlFilename = Server.MapPath("xml") + "\\NavigationBar.xml";
  navBar.navBarName = "Public";
   navBar.backgroundColor  =  ColorTranslator.FromHtml("#6B0808");
   }  // Page_Load
   } // CH05DisplayNavBarCS
}
```

Recipe 5.4. Reusing Code-Behind Classes

Problem

You have several page sections that require identical code-behind but the user presentation must be different for each.

Solution

Create a user control for the first page section. For the other page sections, create only the *.ascx* file and link it to the code-behind class for the first page section. For example, to produce the vertically oriented navigation shown in Figure 5-3, create the navigation bar user control described in Recipe 5.2, then create the *.ascx* file shown in Example 5-14, all without writing any VB or C# code.

Figure 5-3. Reuse of code-behind class output

Discussion

The `@ Control` directive at the top of the *.ascx* page defines the code-behind class that will be used with the *.ascx* file. The `Codebehind` attribute defines the name of the file containing the code-behind class. The `Inherits` attribute defines the class in the codebehind file that inherits from `System.Web.UI.UserControl` and provides the codebehind code.

By changing the `Codebehind` and `Inherits` attributes of the *.ascx* file, you can reuse code. In our example, the attributes are set as shown here to reuse the code-behind from Recipe 5.2:

```
<%@ Control Language="VB" AutoEventWireup="false"
```

```
    CodeFile="CH05UserControlNavBarVB.ascx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH05UserControlNavBarVB" %>
```

The user control can provide any desired user interface without changing any of the code-behind. The server controls referenced in the code-behind class must be present in the*.ascx*, and they must be of the same type. Leaving a server control out of the *.ascx* or changing its type will result in an exception being thrown.

Example 5-14. Reuse of code-behind class (.ascx)

```
<%@ Control Language="VB" AutoEventWireup="false"
    CodeFile="CH05UserControlNavBarVB.ascx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH05UserControlNavBarVB" %>
<asp:datalist id="dlNavBar" runat="server"
      borderwidth="0" cellpadding="0" cellspacing="0" height="29"
      repeatdirection="vertical" repeatcolumns="1"
      repeatlayout="Table" width="150" horizontalalign="Left"
     itemstyle-horizontalalign="Center"
     OnItemDataBound="dlNavBar_ItemDataBound">
 <itemtemplate>
   <a id="anNavBarLink" runat="server" >
    <img id="imgNavBarImage" runat="server" border="0"
     alt="" src=""/></a>
 </itemtemplate>
</asp:datalist>
```

PREV

Recipe 5.5. Communicating Between User Controls

Problem

You have multiple user controls on a page, and one of the user controls needs to send data to another as, for example, when one control takes its form or content from the user's action on another.

Solution

Create a custom event argument class to define the message to be sent, a source user control, a destination user control, and a web form that contains both user controls. (See Recipes 5.1 and 5.2 fo detailed steps.)

In the custom event argument class that defines the message to be sent:

1. Inherit from `EventArgs` .
2. Add a message property to contain the message data.

In the source user control:

1. Create a custom event.
2. Raise the event when the required action is performed, such as when a user completes her entry for a form.

In the destination user control:

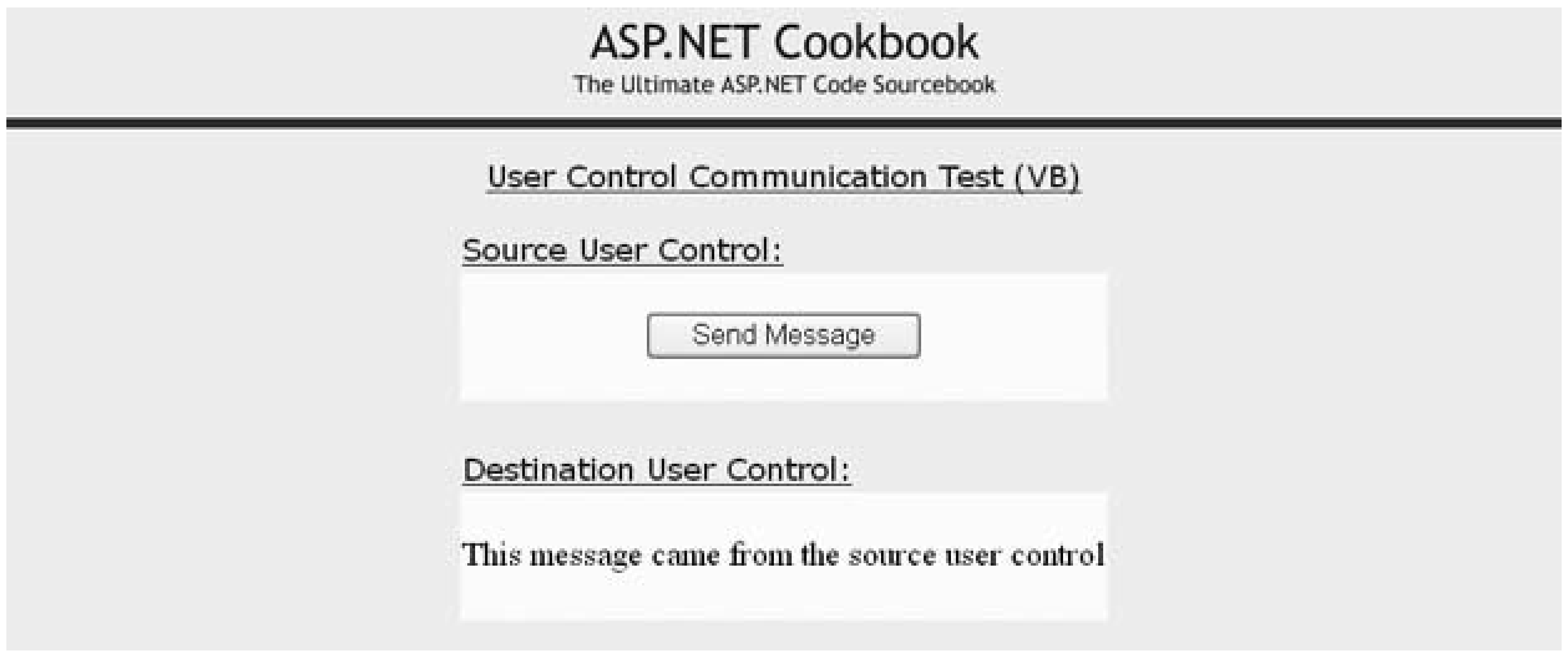
1. Create an event handler to receive the event from the source user control.
2. Display the message received.

In the web form used to test the user control communication:

1. Add the source user control.
2. Add the destination user control.
3. "Wire" the event raised in the source user control to the event handler in the destination user control in the `Page_Load` event of the web form.

The output of a test page showing one user control communicating with another appears in Figure 5-4 . The code for our example application that implements the solution is shown in Examples 5-15, 5-16 , 5-17 , 5-18 , 5-19 , 5-20 , 5-21 , 5-22 , 5-23 , 5-24 through 5-25 . Examples 5-15 (VB) and 5-16 (C#) show the custom event argument class used to define the message sent. Example 5-17 shows the *.ascx* file for the source user control. Examples 5-18 and 5-19 show the VB and C# code-behind for the source user control. Example 5-20 shows the *.ascx* file for the destination user control. Examples 5-21 and 5-22 show the VB and C# code-behind for the destination user control. Example 5-23 shows the *.aspx* file for the web form used to demonstrate the user controls and their inter-connection. Examples 5-24 and 5-25 show the VB and C# code-behind for the demonstration web form.

Figure 5-4. Communicating between user controls output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
User Control Communication Test (VB)
Source User Control:
Send Message
Destination User Control:

This message came from the source user control

Discussion

Rather than dwell on the basic content and creation of user controls, which is the subject of the previous recipes in the chapter, this recipe focuses on the interaction between user controls. The approach we advocate for handling this interaction involves creating a custom event for the source user control and raising the event when the communication is to be initiated, such as when the user clicks a button to complete his entry for the form. To receive the event from the source user control, the destination user control must have an event handler tailored for that purpose.

In our approach, creating the custom event for the source user control involves creating a custom event argument class, which provides the ability to add a message to the event arguments. It involves using a delegate, which is a convenient way to pass to the destination user control a reference to an event handler for the `OnSend` event raised by the source user control.

We've created an application to illustrate our approach. Because of the unusually high number of interrelated files, this example may appear overwhelming at first, but it is actually pretty straightforward. Keep in mind the four basic pieces:

1. A custom event argument class defining the message sent
2. A user control that sends a message (the source)

3. A user control that receives the message (the destination)

4. A web form that contains the two user controls and wires them together

The custom event argument class provides the ability to add the message to the event arguments. This class inherits from `System.EventArgs` and adds a `message` property as shown in Examples 5-15 (VB) and 5-16 (C#).

The source user control contains only a button used to initiate sending a message.

The source user control code-behind contains the bulk of the code. First, we define a new delegate signature, `customMessageHandler` , to allow the `MessageEventArgs` object to be passed as the event arguments. Without this delegate, you would have to use the `EventArgs` object, which does not provide the ability to pass custom information. An event is then defined with this type of delegate.

A *delegate* is a class that can hold a reference to a method. A delegate class has a signature and it can hold references only to methods that match its signature. The delegate object is passed to code that calls the referenced method without having to know at compile time which method will be invoked. The most common example is building a generic sort routine, one that allows you to sort any type of data, where you pass to it the data to be sorted and a reference to the comparison routine needed to compare the particular data. The situation here is somewhat similar. In this case, we are passing a message to the destination user control (contained within an instance of `MessageEventArgs`) and a reference to an event handler for the `OnSend` event raised by the source user control. A delegate provides the best, most convenient way to accomplish this.

Our remaining task in the source user control code-behind is to provide a standard event handler for the send message button click event. In this handler, an instance of `MessageEventArgs` is created and populated with the message being sent. The `OnSend` event is then raised, passing a reference to the source user control as the event source and a reference to the `messageArgs` object containing the message being sent. In our example, this is a hardwired message, but it demonstrates the basic principal.

In C#, the `OnSend` event must be checked to make sure it is not null before raising the event. Failure to do so will result in an exception being thrown if no handler is wired to the event. This is not required for VB.

Our example's destination user control, which is shown in Example 5-20, contains only a label used to display the message sent from the source user control.

The destination user control code-behind, shown in VB in Example 5-21 and in C# in Example 5-22 , contains a single method to handle the event raised from the source user control. The signature of the method must match the `customMessageHandler` delegate defined in the source user control. The only operation performed is to update the label in the user control with the message passed in the event arguments.

In our example, the *.aspx* file for the web form used to demonstrate the user controls, which appears in Example 5-23 , includes the registration of the two user controls and the tags within the HTML where the user controls are to be displayed.

The code-behind for the demonstration web form, shown in VB in Example 5-24 and in C# in Example 5-25 , provides the glue for tying the event from the source user control to the destination user control. This is done by adding the `updateLabel` of the destination user control as an event handler for the `OnSend` event raised by the source user control. We are adding a delegate to the source user control's `OnSend` event's event handler list; that list now consists of just one event handler but can include more.

> Event delegates in .NET are multicast, which allows them to hold references to more than one event handler. This provides the ability for one event to be processed by multiple event handlers. You can try it yourself by adding a label to the demonstration web form, adding a new event handler in the web form, and then adding your new event handler to the `OnSend` event's event handler list. This will cause the label on the destination user control and the web form to be updated with the message from the source user control. An example that does this with multiple user controls is shown in Recipe 5.5.

> In VB, when using the event/delegate model, the keyword `WithEvents` is not used. (The `WithEvents` keyword indicates that a declared object variable refers to a class instance that can raise events.) `WithEvents` and the event/delegate model can be intermixed, but they should not be used for the same event.

See Also

Recipe 5.5 and *Programming C#* or *Programming Visual Basic .NET* (O'Reilly), both by Jesse Liberty, for more about delegates

Example 5-15. Class defining message passed between user controls (.vb)

```
Option Explicit On
Option Strict On

Imports System
Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class class provides the definition of the custom event arguments
  ''' used as the event arguments for the message sent from this control
  ''' This class simply inherits from System.EventArgs and adds a message
  ''' property
  ''' </summary>
  Public Class MessageEventArgs
    Inherits EventArgs

 Private mMessage As String

 '''**************************************************************************
 ''' <summary>
 ''' This property provides the ability to get/set the message in the
 ''' event args
 ''' </summary>
 Public Property message( ) As String
   Get
     Return (mMessage)
   End Get
   Set(ByVal Value As String)
     mMessage = Value
   End Set
    End Property
  End Class 'MessageEventArgs
End Namespace
```

Example 5-16. Class defining message passed between user controls (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the definition of the custom event arguments used
  /// as the event arguments for the message sent from this control. This
  /// class simply inherits from System.EventArgs and adds a message property.
  /// </summary>
  public class MessageEventArgs
  {
    private String mMessage;

 /// <summary>
 /// This property provides the ability to get/set the message in the
 /// event args
 /// </summary>
 public String message
 {
   get
   {
  return (mMessage);
   }

   set
   {
     mMessage = value;
   }
 } // message
  } // MessageEventArgs
}
```

Example 5-17. Communicating between controlssource user control (.ascx

```
<%@ Control Language="VB" AutoEventWireup="false"
      CodeFile="CH05UserControlCommSourceVB.ascx.vb"
   Inherits="ASPNetCookbook.VBExamples.CH05UserControlCommSourceVB"  %>
<asp:Button ID="btnSendMessage" runat="server"
      Text="Send Message"
      OnClick="btnSendMessage_Click" />
```

Example 5-18. Communicating between controlssource user control (.vb)

```
Option Explicit On
Option Strict On

Imports System.Drawing

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code-behind for
  '''  CH05UserControlCommSourceVB.ascx
  ''' </summary>
  Partial Class CH05UserControlCommSourceVB
    Inherits System.Web.UI.UserControl

 'define the delegate handler signature and the event that will be raised
 'to send the message
 Public Delegate Sub customMessageHandler(ByVal sender As System.Object, _
         ByVal e As MessageEventArgs)
 Public Event OnSend As customMessageHandler

 '''*********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the send message button
 ''' click event. It creates a new MessageEventArgs object then raises
 ''' an OnSend event
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnSendMessage_Click(ByVal sender As Object, _
          ByVal e As System.EventArgs)

   Dim messageArgs As New MessageEventArgs
   messageArgs.message = "This message came from the source user control"
   RaiseEvent OnSend(Me, messageArgs)
      End Sub 'btnSendMessage_Click
    End Class 'CH05UserControlCommSourceVB
End Namespace
```

Example 5-19. Communicating between controlssource user control (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
    /// <summary>
 /// This class provides the code-behind for
 ///  CH05UserControlCommSourceCS.ascx
 /// </summary>
 public partial class CH05UserControlCommSourceCS : System.Web.UI.UserControl
 {
   // define the delegate handler signature and the event that will be raised
   // to send the message
   public delegate void customMessageHandler(Object sender,
         MessageEventArgs e);
   public event customMessageHandler OnSend;


   ///*********************************************************************
   /// <summary>
   /// This routine provides the event handler for the send message button
   /// click event. It creates a new MessageEventArgs object then raises
   /// an OnSend event
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void btnSendMessage_Click(object sender,
          EventArgs e)
   {
     MessageEventArgs messageArgs = new MessageEventArgs();
  messageArgs.message = "This message came from the source user control";

  if (OnSend != null)
  {
    OnSend(this, messageArgs);
  }
  }  // btnSendMessage_Click
   }  // CH05UserControlCommSourceCS
}
```

Example 5-20. Communicating between controlsdestination user control (.ascx)

```
<%@ Control Language="VB" AutoEventWireup="false"
    CodeFile="CH05UserControlCommDestinationVB.ascx.vb"
   Inherits="ASPNetCookbook.VBExamples.CH05UserControlCommDestinationVB"  %>
x<asp:Label ID="labMessage" Runat="server">No Message Yet</asp:Label>
```

Example 5-21. Communicating between controlsdestination user control (.vb)

```
Option Explicit On
Option Strict On

Imports System

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH05UserControlCommDestinationVB.ascx
 ''' </summary>
 Partial  Class  CH05UserControlCommDestinationVB
  Inherits System.Web.UI.UserControl
  '''*****************************************************************************
  ''' <summary>
  ''' This routine provides the event handler that is the recipient of the
  ''' event raised by the source user control.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Public Sub updateLabel(ByVal sender As System.Object, _
         ByVal e As MessageEventArgs)
  'update the label with the message in the event arguments
  labMessage.Text = e.message
 End Sub 'updateLabel
 End  Class  'CH05UserControlCommDestinationVB
End Namespace
```

Example 5-22. *Communicating between controlsdestination user control (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
   ///  CH05UserControlCommDestinationCS.ascx
  /// </summary>
   public partial class CH05UserControlCommDestinationCS :
     System.Web.UI.UserControl
  {
     ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler that is the recipient of the
 /// event raised by the source user control.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 public void updateLabel(Object sender,
     MessageEventArgs e)
 {
 // update the label with the message in the event arguments
 labMessage.Text = e.message;
 } // updateLabel
   } // CH05UserControlCommDestinationCS
}
```

Example 5-23. Communicating between controlsmain form (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master";
  AutoEventWireup="false"
   CodeFile="CH05UserControlCommTestVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH05UserControlCommTestVB"
  Title="User Control Communication Test" %>
<%@ Register TagPrefix="ASPCookbook" TagName="SourceControl"
    Src="CH05UserControlCommSourceVB.ascx" %>
<%@ Register TagPrefix="ASPCookbook" TagName="DestinationControl"
    Src="CH05UserControlCommDestinationVB.ascx" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
   <div align="center" class="pageHeading">
        User Control Communication Test (VB)
      </div>
    <table width="60%" align="center" border="0">
       <tr>
   <td class="PageHeading">
```

```
     Source User Control:
   </td>
    </tr>
   <tr>
    <td bgcolor="#ffffcc" align="center" height="75">
       <ASPCookbook:SourceControl id="ucSource" runat="server" />
   </td>
   </tr>
   <tr>
   <td>&nbsp;</td>
   </tr>
   <tr>
  <td class="PageHeading">
    Destination User Control:
  </td>
   </tr>
 <tr>
  <td bgcolor=";#ffffcc" align="center" height="75">
       <ASPCookbook:DestinationControl id="ucDestination" runat="server" />
  </td>
 </tr>
  </table>
</asp:Content>
```

Example 5-24. Communicating between controlsmain form (.vb)

```
Option Explicit On
Option Strict On

Imports System.Drawing

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
   '''  CH05UserControlCommTestVB.aspx
  ''' </summary>
  Partial Class CH05UserControlCommTestVB
    Inherits System.Web.UI.Page
 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
   'wire the event to the destination user control handler
   AddHandler ucSource.OnSend, AddressOf ucDestination.updateLabel
 End Sub  'Page_Load
  End Class  'CH05UserControlCommTestVB
End Namespace
```

Example 5-25. Communicating between controlsmain form (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
  ///  CH05UserControlCommTestCS.aspx
  /// </summary>
  public partial class CH05UserControlCommTestCS : System.Web.UI.Page
  {
 ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
   // wire the event to the destination user control handler
ucSource.OnSend +=
   new  CH05UserControlCommSourceCS.customMessageHandler(ucDestination.updateLabel);
 } // Page_Load
  } // CH05UserControlCommTestCS
}
```

Recipe 5.6. Adding User Controls Dynamically

Problem

You need to load a group of user controls programmatically at runtime because the number of controls required is unknown at design time.

Solution

Bind your data to a `Repeater` control in the normal fashion and then, as data is bound to each row of the `Repeater`, use the ItemDataBound event to load a user control dynamically and place it in a table cell of the `Repeater` control's `ItemTemplate`.

Add a `Repeater` control to the *.aspx* file with a table cell in the `ItemTemplate` where the user control is to be placed.

In the code-behind class, use the .NET language of your choice to:

1. Bind the data to the `Repeater` control.
2. Create an event handler method for the `ItemDataBound` event of the `Repeater` control.
3. In the method that handles the `ItemDataBound` event, use the `LoadControl` method to create an instance of the user control, and then add the loaded control to the controls collection of the table cell in the `ItemTemplate`.

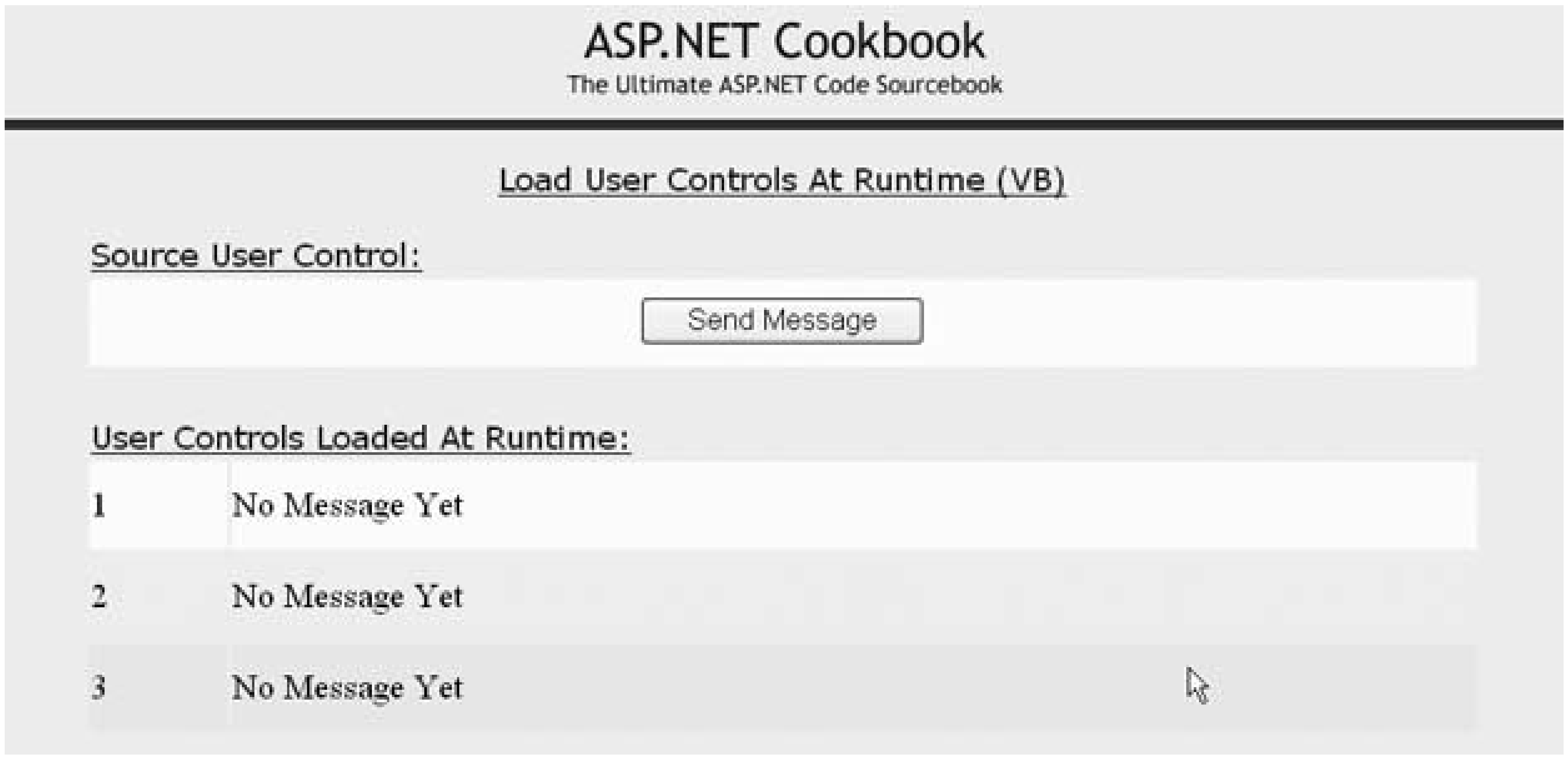
Figure 5-5 shows a form where we start with the user controls created in Recipe 5.4 and dynamically load three user controls at runtime. Example 5-26 shows the *.aspx* file that implements this solution, while Examples 5-27 and 5-28 show the companion VB and C# code-behind files.

Discussion

This recipe demonstrates how to load a group of user controls dynamically into a form, the count for which can be determined only at runtime. A `Repeater` control is used because it generates a lightweight read-only tabular display and is templatedriven. The `Repeater` control's `ItemTemplate` element formats the rows of data. The user control dynamically loaded at runtime is strategically placed in a table cell in the `ItemTemplate`. This loading takes place in the method that handles the `ItemDataBound` event for each row of the `Repeater`. More specifically, the `LoadControl` method is used to create an instance of the user control, and then the loaded control is added to the controls collection of the table cell.

The example we have written to demonstrate the solution starts with the user controls created in Recipe 5.4 and loads the destination user controls at runtime. In addition, it wires them to the source user control to demonstrate the multicast event mechanism in .NET.

Figure 5-5. User controls loaded at runtime output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Load User Controls At Runtime (VB)

Source User Control:

User Controls Loaded At Runtime:

1	No Message Yet
2	No Message Yet
3	No Message Yet

A `Repeater` control is placed in the *.aspx* file with an `ItemTemplate` containing two table cells. The first cell is used to hold the dynamically loaded user control's number, and the second cell is used to hold the user control itself. Example 5-26 shows how we've implemented this in our example.

> The dynamically loaded user controls can beadded to the `Page` control collection; however, this will place them at the bottom of the page and they will be rendered outside the form. Dynamically loaded user controls should be added to the controls collection of some control contained within the form.

In the `repUserControls_ItemDataBound` method of the code-behind, the user control for the row being bound is loaded at runtime from the *.ascx* file using the `LoadControl` method. It is then added to the controls collection of the second table cell in the`Repeater`.

To demonstrate the multicast event mechanism in .NET we mentioned in Recipe 5.4, each of the dynamically loaded user controls is wired to the source user control in the*.aspx* file. This results in each of the dynamically loaded user controls receiving the message event from the source user control.

```
AddHandler ucSource.OnSend, AddressOf ucDest.updateLabel
```

```
ucSource.OnSend +=
   new  CH05UserControlCommSourceCS.customMessageHandler(ucDest.updateLabel);
```

The result in this case is that each destination user control is updated with the same text from the source user control, which is a bit dull. Imagining a more interesting scenario is easy where one destination user control has a text label updated, the second a database, and the third an XML web service, or the like, with all of these updates the result of methods having been registered with the source control's `OnSend` event's event handler list.

See Also

Recipe 5.4

Example 5-26. User controls loaded at runtime (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH05UserControlRuntimeVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH05UserControlRuntimeVB"
 Title="Load User Controls At Runtime" %>
<%@ Register TagPrefix="ASPCookbook" TagName="SourceControl"
     Src="CH05UserControlCommSourceVB.ascx" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Load User Controls At Runtime (VB)
 </div>
 <table width="90%" align="center" border="0" >
  <tr>
     <td class="PageHeading" colspan="2">
    Source User Control:</td>
  </tr>
  <tr>
    <td bgcolor="#ffffcc" align="center" height="50" colspan="2">
    <ASPCookbook:SourceControl id="ucSource" runat="server" />
   </td>
  </tr>
  <tr>
    <td colspan="2">&nbsp;</td>
  </tr>
  <tr>
    <td class="PageHeading" colspan="2">
    User Controls Loaded At Runtime:
   </td>
  </tr>
  <asp:repeater id="repUserControls" runat="server"
      OnItemDataBound="repUserControls_ItemDataBound">
   <itemtemplate>
    <tr id="trControl" runat="server" height="50">
    <td id="tdCount" runat="server" width="10%"></td>
    <td id="tdUserControl" runat="server"></td>
    </tr>
   </itemtemplate>
  </asp:repeater>
 </table>
</asp:Content>
```

Example 5-27. User controls loaded at runtime (.vb)

```
Option Explicit On
Option Strict On

Imports System.Web.UI.WebControls

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH05UserControlRuntimeVB.aspx
 ''' </summary>
 Partial Class CH05UserControlRuntimeVB
  Inherits System.Web.UI.Page
  'the following variable is used to keep count of the number of controls
  Private controlCount As Integer

  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
   Dim values As ArrayList = New ArrayList

   'build array of data to bind to repeater
   'for this example it is just the color of the entry but for a real
   'application the data would normally be from a database, etc.
   values.Add("#ffffcc")
   values.Add("#ccffff")
    values.Add("#ccff99")
   'bind the data to the repeater
   controlCount = 0
   repUserControls.DataSource = values
   repUserControls.DataBind( )
  End Sub 'Page_Load

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the item data bound event
  ''' of the repeater control on the form. It is responsible for loading
  ''' the user control and placing it in the repeater for the item being
  ''' bound.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub repUserControls_ItemDataBound(ByVal sender As Object, _
          ByVal e As RepeaterItemEventArgs)
   'the following constants are the names of the controls in the repeater
   Const TABLE_ROW As String = "trControl"
   Const COUNT_CELL As String = "tdCount"
   Const USER_CONTROL_CELL As String = "tdUserControl"

   Dim row As HtmlTableRow
   Dim cell As HtmlTableCell
   Dim ucDest As CH05UserControlCommDestinationVB

   'make sure this is an item or alternating item in the repeater
   If ((e.Item.ItemType = ListItemType.Item) Or _
    (e.Item.ItemType = ListItemType.AlternatingItem)) Then
    'find the table row and set the background color
    row = CType(e.Item.FindControl(TABLE_ROW), _
      HtmlTableRow)
    row.BgColor = CStr(e.Item.DataItem)

    'find the cell for the control count and set the count
    cell = CType(e.Item.FindControl(COUNT_CELL), _
       HtmlTableCell)
    controlCount += 1
    cell.InnerText = controlCount.ToString()

    'find the cell for the control and load a user control
    cell = CType(e.Item.FindControl(USER_CONTROL_CELL), _
     HtmlTableCell)
    ucDest = CType(LoadControl("CH05UserControlCommDestinationVB.ascx"), _
    CH05UserControlCommDestinationVB)
    cell.Controls.Add(ucDest)
    AddHandler ucSource.OnSend, AddressOf ucDest.updateLabel
   End If
  End Sub  'repUserControls_ItemDataBound
 End Class 'CH05UserControlRuntimeVB
End Namespace
```

Example 5-28. User controls loaded at runtime (.cs)

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

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH05UserControlRuntimeCS.aspx
 /// </summary>
 public partial class CH05UserControlRuntimeCS : System.Web.UI.Page
 {
  // the following variable is used to keep count of the number of controls
  private int controlCount;
  ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   ArrayList values = new ArrayList();

   // build array of data to bind to repeater
   // for this example it is just the color of the entry but for a real
   // application the data would normally be from a database, etc.
   values.Add("#ffffcc");
   values.Add("#ccffff");
   values.Add("#ccff99");

   // bind the data to the repeater
   controlCount = 0;
   repUserControls.DataSource = values;
   repUserControls.DataBind();
  } // Page_Load

  ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the item data bound event
  /// of the repeater control on the form. It is responsible for loading
  /// the user control and placing it in the repeater for the item being
  /// bound.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void repUserControls_ItemDataBound(Object sender,
         RepeaterItemEventArgs e)
  {
   // the following constants are the names of the controls in the repeater
   const String TABLE_ROW = "trControl";
   const String COUNT_CELL = "tdCount";
   const String USER_CONTROL_CELL = "tdUserControl";

   HtmlTableRow row;
   HtmlTableCell cell;
    CH05UserControlCommDestinationCS  ucDest;

   // make sure this is an item or alternating item in the repeater
   if ((e.Item.ItemType == ListItemType.Item) ||
    (e.Item.ItemType == ListItemType.AlternatingItem))
   {
    // find the table row and set the background color
    row = (HtmlTableRow)(e.Item.FindControl(TABLE_ROW));
    row.BgColor = (String)(e.Item.DataItem);

    // find the cell for the control count and set the count
    cell = (HtmlTableCell)(e.Item.FindControl(COUNT_CELL));
    controlCount += 1;
    cell.InnerText = controlCount.ToString();

    // find the cell for the control and load a user control
    cell = (HtmlTableCell)(e.Item.FindControl(USER_CONTROL_CELL));
    ucDest = (CH05UserControlCommDestinationCS)
    (LoadControl("CH05UserControlCommDestinationCS.ascx"));
    cell.Controls.Add(ucDest);
    ucSource.OnSend +=
    new
      CH05UserControlCommSourceCS.customMessageHandler(ucDest.updateLabel);
   }
  } // repUserControls_ItemDataBound
 } //  CH05UserControlRuntimeCS
}
```

PREV

Chapter 6. Custom Controls

PREV

6.0 Introduction

Custom controls are compiled controls that function like ASP.NET's own server controls. Like user controls, custom controls can enhance the reusability of code repeated within a project or over multiple projects. There are important differences, however. In general, user controls are like "mini" web pages in that they contain part of an ASP.NET page. Additionally, they are compiled when first requested, but their user interface can easily be changed as required. With custom controls, on the other hand, the user interface is generated by the code and cannot easily be changed except through properties and methods that you implement, about which we'll speak more in a minute.

In a broader sense, a custom control is any control you create with these common themes: it is typically derived from the `Control` or `WebControl` class in the `System.Web.UI` namespace or an existing ASP.NET server control. It generally provides its own user interface, and it may provide its own backend functionality through the methods, properties, and events that you implement for it.

Custom controls range from the simple to the complex. A simple custom control might, for example, write some HTML, perhaps modifying its HTML-style attributes as it does so. A more complex custom control would offer HTML-style attributes of its own through properties you implement. A custom text box control could, for example, offer one attribute for controlling the color of its label and another to control the width of the control itself. To make it more complex and more useful, the control would have to handle `postback` events like the server controls provided with ASP.NET. Still more complex would be a custom control that, like the `DataGrid` control, includes templates and supports data binding. Because custom controls can be created from scratch or inherited from existing controls, the possibilities are endless.

All of the hypothetical custom controls we've described are, with the exception of the `templated` control, illustrated in this chapter's recipes. (If you're interested in learning more about `templated` controls, see the example in the *ASP.NET QuickStart Tutorials* that are part of the *.NET Framework QuickStarts* that ship with Visual Studio or are available via http://www.gotdotnet.com.)

This chapter introduces you to some of the techniques used to build customcontrols. In sticking to the basics, we implicitly recognize that custom controls are "custom" and, therefore, highly individual However, these basics ought to take you a long way in crafting your own custom controls.

Which Is Better: Control or WebControl?

The Control class in the System.Web.UI namespace is the base class for all server controls. It provides the properties, methods, and events shared by all web controls. The WebControl class in the System.Web.UI.WebControls namespace derives from the Control class and adds style properties such as Font, Forecolor, and Backcolor. In addition, it provides skin and theme features.

Microsoft recommends deriving from the Control class if your custom control contains no user interface elements. But if your custom control provides a user interface, you should derive from the WebControl class.

PREV

PREV

Recipe 6.2. Combining HTML Controls in a Single Custom Control

Problem

You want to create a custom control that combines two or more HTML controls.

Solution

Use the .NET language of your choice to:

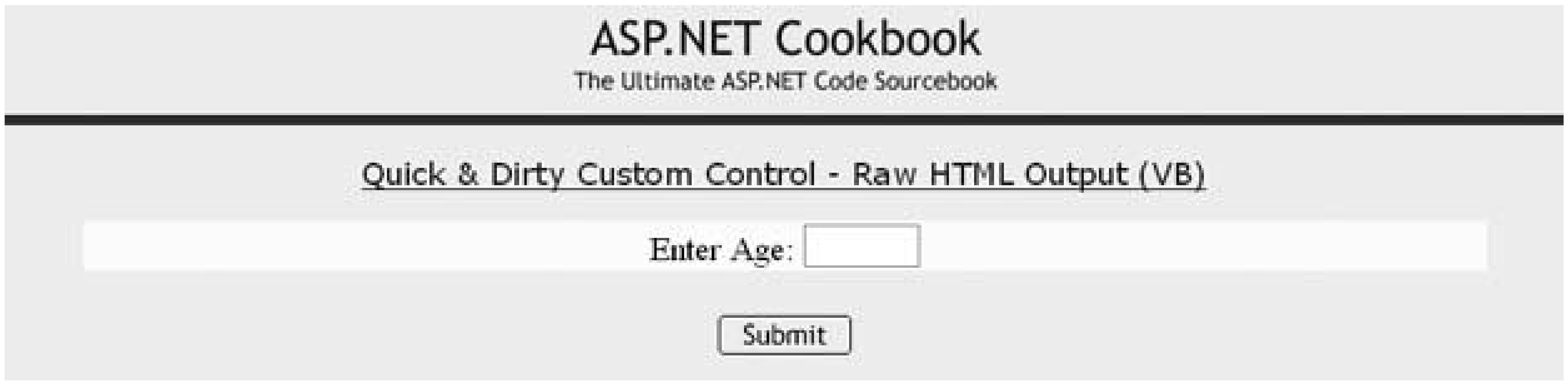
1. Create a class that inherits from the `WebControl` class in the `System.Web.UI.WebControls` namespace.
2. Override the `Render` method to have it output the HTML controls you wish to include.
3. (Optional) Use the `HtmlTextWriter` class to enhance your chances of writing well-formed HTML.

To use the custom control in an ASP.NET page:

1. Register the assembly containing the control.
2. Insert the tag for the custom control anywhere in the page.

Figure 6-1 shows the output of a custom control that combines a label and text box. Examples 6-1 and 6-2 show the VB and C# class files for the custom control. Example 6-3 shows how to use the custom control in an ASP.NET page.

Figure 6-1. Basic custom control output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Quick & Dirty Custom Control - Raw HTML Output (VB)
Enter Age:
Submit

Discussion

To create a custom control that combines the functionality of two or moreHTML controls, you first create a class that inherits from the WebControl class in System.Web.UI.WebControls.

The only method of WebControl required to output HTML is the Render method. Render is responsible for writing the HTML that will be rendered by the browser. To enhance your ability to write well-formed HTML, you can use other methods of the HtmlTextWriter class along with the HtmlTextWriterAttribute and HtmlTextWriterTag enumerations. We'll talk more about this in a minute, but for now we'll stick with writing our own unvarnished HTML.

The custom control we have implemented in our example contains a label and an input control. The label and input control are output in the Render method with the following code:

```
writer.Write("Enter Age: ")
writer.Write("<input type='text' size='3' />")
```

```
writer.Write("Enter Age: ");
writer.Write("<input type='text' size='3' />");
```

To use the custom control, the assembly containing the control must be registered in the target *.aspx* file. The TagPrefix attribute defines an alias to use for the namespace in the page. TheNamespace attribute must be set to the fully qualified namespace of the control. Here is how you register the assembly in our example:

```
<%@ Register TagPrefix="ASPCookbook" Namespace="ASPNetCookbook.VBExamples"; %>
```

If you place the source code for your custom controls in the App_Code folder of your project, ASP.NET 2.0 will dynamically compile the code into an assembly. If you do not include the Assembly attribute in the @ Register directive, the assembly created by compiling the code in the App_Code folder will be inferred.

If you precompile your source code into an explicit assembly, the Assembly attribute of the @ Register directive must be included to inform ASP.NET where to locate the custom control.

The custom control can be placed anywhere on the page by inserting a tag. The control to insert is identified by naming the tag with the TagPrefix followed by the class name. The tag must include the id and runat="server" attributes for the control to be rendered on the page. This is the tag used in our example:

```
<ASPCookbook:CH06QuickAndDirtyCustomControlVB1
   id="ccQuickAndDirty" runat="server" />
```

In our example, raw HTML is written to the web page in the Render method. For simple HTML, this works well. As the complexity of the HTML you write increases, however, the likelihood that you'll introduce errors increases. Fortunately, HtmlTextWriter includes methods that simplify the generation of complex HTML. These methods can help you with the nuances of adding HTML attributes (and values) to an HTMLTextWriter output stream, writing beginning and ending tags, flushing buffers so all buffered data is written to the text stream, etc. To create the input box in our example, you could use the HtmlTextWriter like this:

```
Protected Overrides Sub Render(ByVal writer As HtmlTextWriter)
 'output label
 writer.Write("Enter Age: ")

 'output input control
 writer.AddAttribute(HtmlTextWriterAttribute.Type, _
   "text")
 writer.AddAttribute(HtmlTextWriterAttribute.Size, _
   "3")
 writer.RenderBeginTag(HtmlTextWriterTag.Input)
 writer.RenderEndTag()
End Sub 'Render
```

```
protected override void Render(HtmlTextWriter writer)
{
 //output label
 writer.Write("Enter Age: ");

 //output input control
```

```
  writer.AddAttribute(HtmlTextWriterAttribute.Type,
    "text");
  writer.AddAttribute(HtmlTextWriterAttribute.Size,
    "3");

  writer.RenderBeginTag(HtmlTextWriterTag.Input);
  writer.RenderEndTag();
} // Render
```

One advantage of implementing the `Render` method in this way is that you can use the `RenderBeginTag` and `RenderEndTag` methods to output HTML and sidestep having to insert the <, /, and > characters yourself. In addition, using the `HtmlTextWriterTag` and `HtmlTextWriterAttribute` enumerations ensures all tags and attributes are correctly spelled.

Another advantage is that you can avoid the hassle of ensuring the single and double quotes are handled correctly. Notice in the first example that the values for the attributes of the input tag were output with single quotes, since double quotes mark the beginning and end of strings. Outputting double quotes around the values would have required more complex code with string concatenations Using the `AddAttribute` method avoids this problem completely.

You must call the `AddAttribute` method immediately before you call the `RenderBeginTag` method, which writes the opening tag of the associated HTML element. This is required because the `HtmlTextWriter` builds a collection of attributes to output in the opening tag of the HTML element. When the `RenderBeginTag` method is called, the attributes are output in the opening tag and then the collection is cleared.

A useful enhancement to this approach would be to add HTML-style attributes to the control to make the control more adaptable and reusable throughout your applications. See Recipe 6.2 for how to do this.

If you are going to use your custom controls on multiple pages in your application and would like to avoid having to place the `@ Register` directive in each page, register the controls in the *web.config* file. To register the custom control for this recipe, for example, the following would be added to the *web.config* file:

```
<system.web>

 <!--
  Register controls that are to be used on multiple
  pages to eliminate the need to add an @Register
  directive to each page.
  -->
 <pages>
  <controls>
   <add tagPrefix="ASPCookbookControls"
   namespace="ASPNetCookbook.VBExamples" />
  </controls>
 </pages>
</system.web>
```

See Also

Recipe 6.2; for additional details on `Control`, `Render`, `HtmlTextWriterTag`, `HtmlText-WriterAttribute`, and especially `HtmlTextWriter`, search the MSDN Library

Example 6-1. Quick-and-dirty custom control (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the quick and dirty custom control example which
 ''' includes a label and a text box. The Control is rendered by writing
 ''' raw HTML to the output stream.
 ''' </summary>
 Public Class CH06QuickAndDirtyCustomControlVB1
  Inherits WebControl

  '''*************************************************************************
  ''' <summary>
  ''' This routine renders the HTML output of the control
  ''' </summary>
  '''
  ''' <param name="writer">Set to the HtmlTextWriter to use to output the
  ''' rendered HTML for the control
  ''' </param>
  Protected Overrides Sub Render(ByVal writer As HtmlTextWriter)
   'output label
   writer.Write("Enter Age: ")

   'output input control
   writer.Write("<input type='text' size='3' />")
  End Sub 'Render
 End Class 'CH06QuickAndDirtyCustomControlVB1
End Namespace
```

Example 6-2. Quick-and-dirty custom control (.cs)

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

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the quick and dirty custom control example which
 /// includes a label and a text box. The Control is rendered by writing
 /// raw HTML to the output stream.
 /// </summary>
 public class CH06QuickAndDirtyCustomControlCS1 : WebControl
 {
  ///*********************************************************************
  /// <summary>
  /// This routine renders the HTML output of the control
  /// </summary>
  ///
  /// <param name="writer">Set to the HtmlTextWriter to use to output the
  /// rendered HTML for the control
  /// </param>
  protected override void Render(HtmlTextWriter writer)
  {
   // output label
   writer.Write("Enter Age: ");

   // output input control
   writer.Write("<input type='text' size='3' />");
  } // Render
 } //  CH06QuickAndDirtyCustomControlCS1
}
```

Example 6-3. Using the quick-and-dirty custom control

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH06DisplayQuickAndDirtyControlVB1.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH06DisplayQuickAndDirtyControlVB1"
 Title="Quick And Dirty Custom Control" %>
<%@ Register TagPrefix="ASPCookbook" Namespace="ASPNetCookbook.VBExamples" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Quick & Dirty Custom Control - Raw HTML Output (VB)
 </div>
 <table width="90%" align="center" border="0">
  <tr bgcolor="#ffffcc">
   <td align="center">
    <ASPCookbook:CH06QuickAndDirtyCustomControlVB1
      id="ccQuickAndDirty" runat="server' />
   </td>
  </tr>
  <tr>
   <td align="center">
    <br/>
    <asp:Button ID="btnSubmit" runat="server"
    Text="Submit" />
   </td>
  </tr>
 </table>
</asp:Content>
```

PREV

Recipe 6.3. Creating a Custom Control with Attributes

Problem

You want to create a custom control with HTML-style attributes that can be used to customize the appearance of the control in the *.aspx* file.

Solution

Create the basic custom control (as described in Recipe 6.1), add properties to the class, and use the values of the properties when rendering the control's HTML output.

Use the .NET language of your choice to:

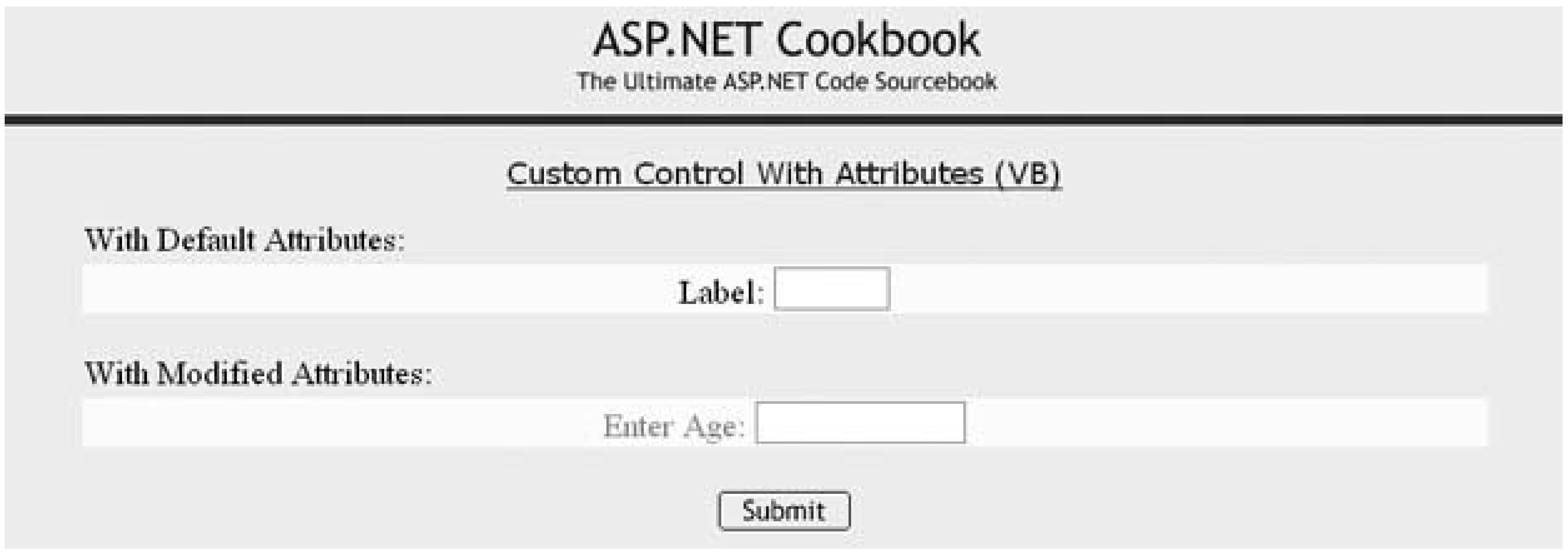
1. Create a class that inherits from the `WebControl` class in the `System.Web.UI.WebControls` namespace.
2. Implement support for the HTML-style attributes by adding properties to the class.
3. Override the `TagKey` property to return the HTML tag to be used as a container for the control.
4. Override the `RenderContents` method to have it render the HTML output of the control using the values of the properties.

To use the custom control in an ASP.NET page:

1. Register the assembly containing the control.
2. Insert the tag for the custom control anywhere in the page and set the attributes appropriately.

To illustrate this solution, we started with the sample custom control we built for Recipe 6.1 and added support for HTML-style attributes, such as an attribute that defines the color used to display label text. Figure 6-2 shows some output using default and modified attributes for the control. In the case of the latter, we used the Enter Age: label text appears in red when rendered on the screen. Examples 6-4 and 6-5 show the VB and C# class files for our custom control. Example 6-6 shows how to use the custom control in an ASP.NET page to produce these results.

Figure 6-2. Custom control with attributes output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Custom Control With Attributes (VB)
With Default Attributes:
Label:
With Modified Attributes:
Enter Age:
Submit

Discussion

Recipe 6.1 describes how to create a basic custom control, so we'll skip that discussion here. Instead, we'll focus on implementing HTML-style attributes for a custom control.

Custom control properties provide the ability to change aspects of the control programmatically using HTML-style attributes, which can be set in the *.aspx* file or code-behind. Attributes are a common feature of ASP.NET server controls. For example, the image button control provides an `ImageURL` attribute that you can set to define the image to be displayed when the control is rendered:

```
<asp:ImageButton ID="btnSubmit"; Runat="server"
   ImageUrl="button_submit.gif" />
```

Attributes are implemented in a custom control by adding properties to the class. The properties are similar to properties in a class implemented as a data service. The names of the properties define the names of the attributes that can be used with the custom control.

To illustrate this approach, we've provided the code in Examples 6-4 (VB) and 6-5 (C#) that defines a custom control named `CH06CustomControlAttributesVB` or `CH06-CustomControlAttributesCS` with the properties shown in Table 6-1.

Table 6-1. Custom control properties

Property	Data type	Description
`labelText`	`String`	Defines the text for the label
`textColor`	`Color`	Defines the color used to display the label text
`textboxWidth`	`Int` or `Integer`	Defines the width of the text box

When you define custom control properties, you should always initialize each property by assigning it a default value. For example, in Examples 6-4 and 6-5, we've assigned the private variable used to store the `labelText` property an initial value of `"Label:"`, the private variable used to store the `textColor` property value has been given an initial value of `Color.Black`, and the private variable used to store the `textboxWidth` property value has been given an initial value of 3. Initializing properties in this manner allows your control to handle the condition when the programmer does not include a value for the attribute that corresponds to a particular property.

To use a custom control, it must first be registered in the target *.aspx* file, as described in Recipe 6.1. For instance, here's how to register the VB version of the custom control in our example:

```
<%@ Register TagPrefix="ASPCookbook" Namespace="ASPNetCookbook.VBExamples" %>
```

You can use the control as is if the default label text, text color, and text box width are acceptable:

```
<ASPCookbook:CH06CustomControlAttributesVB
    id="ccAttributes1" runat="server" />
```

Or you can set the attributes to customize the look of the control for a particular page:

```
<ASPCookbook:CH06CustomControlAttributesVB
    id="ccAttributes2" runat="server"
   labelText = "Enter Age: "
   textColor="#ff0000"
   textboxWidth="10"
   CssClass="customControl" />
```

One thing to consider when implementing the properties of a custom control is that ASP.NET will match the names of the attributes in the custom control tag (*.aspx* file) with the names of the properties in the class that implements the custom control. What's more, because all attribute values in the *.aspx* file are strings, ASP.NET provides the type conversion necessary to match the type required for the property.

If ASP.NET cannot convert the attribute value to the type required for the property, a parse exception will be thrown when the page is displayed. For example, if a property is an integer type and the value `"abc"` is set as the attribute value, a parse exception will be thrown. If you set the attributes in the *.aspx* file, there will be no way to prevent this other than to ensure you test the code. Alternately, you can set the values in the code-behind, which will generally catch errors of this type during compilation instead of at runtime.

By default, a custom control that derives from `WebControl` is rendered within an HTML `<span>` tag. You can change the HTML container by overriding the `TagKey` property and returning the desired HTML tag, as we have done in our example where we use a `<div>` tag as the container instead. You'll see in a minute how this plays out in the HTML that is rendered.

VB

```
Protected Overrides ReadOnly Property TagKey() As HtmlTextWriterTag
  Get
    Return (HtmlTextWriterTag.Div)
  End Get
End Property 'TagKey
```

C#

```
protected override HtmlTextWriterTag TagKey
{
  get
  {
    return (HtmlTextWriterTag.Div);
  }
} // TagKey
```

When deriving your custom control from `WebControl`, you should override the `RenderContents` method to output the contents of your control. The `RenderContents` method is called after the `Render` method has rendered the HTML tag (and its attributes) used as the container for your control. By using this approach, any attributes included in the control in the *.aspx* file will be output as attributes of control container. In our example, we have included an attribute for the `CssClass`:

```
<ASPCookbook:CH06CustomControlAttributesVB
    id="ccAttributes2" runat="server"
   labelText = "Enter Age: "
    textColor="#ff0000"
    textboxWidth="10"
   CssClass="customControl" />
```

If you override the `Render` method, the `RenderContents` method will not be called, since the `Render` method sets up the rendering execution life cycle. Generally, you should override only the `RenderContents` method.

When the control is rendered, the following HTML is sent to the browser with the class attribute set t the value of the `CssClass` attribute in the server control:

```
<div id="ctl00_PageBody_ccAttributes2" class="customControl" >
 <font color="Red">Enter Age: </font><input type="text" size="10" />
</div>
```

See Also

Recipe 6.1

Example 6-4. Custom control with attributes (.vb)

```
Option Explicit On
Option Strict On

Imports System.Drawing

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides a custom control with attributes to provide the
 ''' ability to alter the control programmically.
 ''' </summary>
 Public Class CH06CustomControlAttributesVB
  Inherits WebControl

  'private copies of attribute data
  Private mLabelText As String = "Label: "
  Private mTextColor As Color = Color.Black
  Private mTextboxWidth As Integer = 3

  '''**********************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the text of of the label
  ''' in the control
  ''' </summary>

  Public Property labelText() As String
   Get
    Return (mLabelText)
   End Get

   Set(ByVal Value As String)
    mLabelText = Value
   End Set
  End Property 'labelText

  '''**********************************************************************
  ''' <summary>
  ''' This property provides the ability to set the color of the text
  ''' in the control
  ''' </summary>
  Public Property textColor() As Color
   Get
    Return (mTextColor)
   End Get

   Set(ByVal Value As Color)
    mTextColor = Value
   End Set
  End Property 'textColor

   '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the width of the text box
  ''' in the control
  ''' </summary>

  Public Property textboxWidth() As Integer
   Get
    Return (mTextboxWidth)
   End Get

   Set(ByVal Value As Integer)
    mTextboxWidth = Value
   End Set
  End Property 'textboxWidth

   '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to set the HTML tag that is used
  ''' as the container for the control
  ''' </summary>
  Protected Overrides ReadOnly Property TagKey() As HtmlTextWriterTag
   Get
    Return (HtmlTextWriterTag.Div)
   End Get
  End Property 'TagKey

   '''*************************************************************************
  ''' <summary>
  ''' This routine renders the HTML output of the control contents
  ''' </summary>
  '''
  ''' <param name="writer">Set to the HtmlTextWriter to use to output the
  ''' rendered HTML for the control
  ''' </param>
  Protected Overrides Sub RenderContents(ByVal writer As HtmlTextWriter)
   'output label within a font tag
   writer.AddAttribute("color", _
        ColorTranslator.ToHtml(textColor))
```

```
    writer.RenderBeginTag(HtmlTextWriterTag.Font)
    writer.Write(labelText)
    writer.RenderEndTag()

    'output input control
    writer.AddAttribute(HtmlTextWriterAttribute.Type, _
      "text")
    writer.AddAttribute(HtmlTextWriterAttribute.Size, _
      textboxWidth.ToString())
    writer.RenderBeginTag(HtmlTextWriterTag.Input)
    writer.RenderEndTag()
  End Sub    'RenderContents
 End Class    'CH06 CustomControlAttributesVB
End Namespace
```

Example 6-5. Custom control with attributes (.cs)

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

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides a custom control with attributes to provide the
 /// ability to alter the control programmically.
 /// </summary>
 public class CH06CustomControlAttributesCS : WebControl
 {
  // private copies of attribute data
  private String mLabelText = "Label: ";
  private Color mTextColor = Color.Black;
  private int mTextboxWidth = 3;

  ///**********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the text of of the label
  /// in the control
  /// </summary>
  public String labelText
  {
   get
   {
    return (mLabelText);
   }
   set
  {
   mLabelText = value;
  }
} // labelText

///***************************************************************************
/// <summary>
/// This property provides the ability to set the color of the text
/// in the control
/// </summary>
public Color textColor
{
  get
  {
   return (mTextColor);
  }
  set
  {
   mTextColor = value;
  }
} // textColor

///***************************************************************************
/// <summary>
/// This property provides the ability to get/set the width of the text box
/// in the control
/// </summary>
public int textboxWidth
{
  get
  {
   return (mTextboxWidth);
  }
  set
  {
   mTextboxWidth = value;
  }
} // textboxWidth

///***************************************************************************
/// <summary>
/// This property provides the ability to set the HTML tag that is used
/// as the container for the control
/// </summary>
protected override HtmlTextWriterTag TagKey
{
  get
  {
  return (HtmlTextWriterTag.Div);
  }
} // TagKey
```

```
   ///*********************************************************************
  /// <summary>
  /// This routine renders the HTML output of the control contents
  /// </summary>
  ///
  /// <param name="writer">Set to the HtmlTextWriter to use to output the
  /// rendered HTML for the control
  /// </param>
  protected override void RenderContents(HtmlTextWriter writer)
  {
   // output label within a font tag
   writer.AddAttribute("color",
     ColorTranslator.ToHtml(textColor));
   writer.RenderBeginTag(HtmlTextWriterTag.Font);
   writer.Write(labelText);
   writer.RenderEndTag();

   // output input control
   writer.AddAttribute(HtmlTextWriterAttribute.Type,
    "text");
   writer.AddAttribute(HtmlTextWriterAttribute.Size,
     textboxWidth.ToString());
   writer.RenderBeginTag(HtmlTextWriterTag.Input);
   writer.RenderEndTag();
  } // RenderContents
 } // CH06CustomControlAttributesCS
}
```

Example 6-6. Using the custom control with attributes (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH06DisplayControlWithAttributesVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH06DisplayControlWithAttributesVB"
 Title="Display Custom Control With Attributes" %>
<%@ Register TagPrefix="ASPCookbook" Namespace="ASPNetCookbook.VBExamples" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Custom Control With Attributes (VB)
 </div>
 <table width="90%" align="center" border="0">
  <tr>
   <td>With Default Attributes:</td>
  </tr>
  <tr bgcolor="#ffffcc">
   <td align="center">
    <ASPCookbook:CH06 CustomControlAttributesVB
```

```
     id="ccAttributes1" runat="server" />
   </td>
  </tr>
  <tr>
   <td><br />With Modified Attributes:</td>
  </tr>
  <tr bgcolor="#ffffcc">
   <td align="center">
    <ASPCookbook:CH06CustomControlAttributesVB
    id="ccAttributes2" runat="server"
    labelText = "Enter Age: "
    textColor="#ff0000"
    textboxWidth="10"
    CssClass="customControl" />
   </td>
  </tr>
  <tr>
   <td align="center">
    <br/>
    <asp:Button ID="btnSubmit" runat="server"
    Text="Submit" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Recipe 6.4. Creating a Custom Control with State

Problem

You want to create a custom control that remembers its state between `postbacks` of a form, like the server controls provided with ASP.NET.

Solution

Create a custom control like the one described in Recipe 6.2, implement the `IPostBackDataHandler` interface to add the functionality to retrieve the values posted to the server and then update the values in the custom control from the `postback` data.

Use the .NET language of your choice to:

1. Create a class that inherits from the `WebControl` class in the `System.Web.UI.WebControls` namespace.
2. Implement support for HTML-style attributes by adding properties to the class.
3. Implement an `IPostBackDataHandler` as necessary to update the state of the control with the posted data.
4. Override the `RenderContents` method to have it render the HTML output of the control using the values of the properties.

To use the custom control in an ASP.NET page:

1. Register the assembly containing the control.
2. Insert the tag for the custom control anywhere in the page and set the attributes appropriately.

Examples 6-7 and 6-8 show the VB and C# class files for a custom control that maintains state. Example 6-9 shows how we use the custom control in an ASP.NET page.

A version of the custom control that maintains state and provides the added ability to raise an event when the control data has changed is shown in Examples 6-10 (VB) and 6-11 (C#). Examples 6-12, 6-13 through 6-14 show the *.aspx* and code-behind files of an application that uses the custom control with state and provides an event handler for the data-changed event.

Discussion

When implementing a custom control, you will want to ensure it maintains its state between `postbacks` to the server. Otherwise, it will lose its values and the user will have to reset them each time. To see what we mean by this, consider the custom controls discussed in Recipes 6.1 and 6.2; if you implement either of these controls, you will notice that anytime you click the Submit button, the value entered into the text box is lost when the page is redisplayed. This is caused by the control not processing the form data posted back to the server.

To maintain the values in a custom control, the control must implement the `IPostBackDataHandler` interface. The `IPostBackDataHandler` interface requires the implementation of two methods: `LoadPostData` and `RaisePostDataChangedEvent`. The `LoadPostData` method supplies the data posted to the server, which provides the ability to update the state of the control with the posted data. The `RaisePostDataChangedEvent` method provides the ability to raise events if data for the control changes; this method is used to good effect in the second of the two approaches we advocate for this recipe, and we discuss it at the end of this section. The `LoadPostData` and `RaisePostDataChangedEvent` methods are automatically called by ASP.NET when the form is posted back to the server.

The `LoadPostData` method has the following signature:

VB
```
Public Overridable Function LoadPostData(ByVal postDataKey As String, _
       ByVal postCollection As NameValueCollection) As Boolean

public virtual bool LoadPostData(string postDataKey,
      NameValueCollection postCollection)
```

The `postCollection` argument contains the collection of name/value pairs posted to the server. The `postDataKey` parameter provides the "name" of the key value for this control used to access the control's `postback` value.

The `postDataKey` parameter will be set to the unique ID value for the custom control. If the custom control contains only one control that posts a value back to the server and its ID was used as the value of the `name` attribute when the control was rendered, the value of the `postDataKey` parameter can be used to obtain the posted value. If the custom control is a composite control that contains more than one control with `postback` data or if the custom control's ID is not used for the control within the custom control, you, as the programmer, will need to provide a unique ID value for each control in the custom control, extract the values individually within the `LoadPostData` method, and set the appropriate property.

The return value for the `LoadPostData` method should always be set to `False` if the data has not changed or if no check is performed to see if the data changed. Setting the return to `False` prevents the `RaisePostDataChangedEvent` method from being called. See the discussion at the end of this section regarding raising events on data changes.

To give you a better feel for how to implement this solution, we turn to a variation of the custom control we've been working with throughout the chapter, which contains a label and text box. In our first example, the control needs to remember the state of the text in the text box of the input HTML control, and so we have added a `text` property to provide the ability to get/set the text value. Similar properties are provided for the label text, text color, and text box width. No surprise here.

As you look over the code, you'll notice that, with the exception of the `text` property, all the properties in this custom control use private variables to store their values. These values are lost when the form is rendered and sent to the browser. Because these values are not changed by the user in the browser, there is no need to remember their previous values. The `text` property is another matter. To provide the ability to remember its previous value, we use the `ViewState` to persist the value instead of a private variable. We have implemented the `LoadPostData` method to process the data posted back from the client and set the `text` property from the `postback` data.

We have also implemented the `RaisePostDataChangedEvent` method. In this first example, the method is empty and not used but is required as part of the `IPostBackDataHandler` interface. In the recipe's second example, it is used to raise an event when data for the text control change (more about this later).

A couple of changes from Recipe 6.2's implementation are required in the `RenderContents` event handler. You must set the `name` and `value` attributes of the HTML input control. Set the `name` attribute to a unique identifier so you'll have the ability to obtain the value posted back to the server. If the control does not have a `name` attribute, the browser will not include its data in the `postback` data. Set the `value` attribute to the current text value for the control.

> If the text value does not exist, the `value` attribute should not be output. Absent this check, the `value` attribute will be output without a value, which is not well-formed HTML and is not handled well by some browsers.

In our second example, we build on the basic structure of the first solely to raise an event if the text in the text box changes and to notify the user with a change in the label text. Because the value that was output when the page was rendered is stored in the `ViewState`, we can compare it to the new value posted back to the server. As shown in Examples 6-10 (VB) and 6-11 (C#), if the value changes, the `LoadPostData` method will return `TRue`. Otherwise, it will return `false`.

When the `LoadPostData` method returns True, ASP.NET will call the `RaisePostDataChangedEvent` method after all of the controls have had the opportunity to process their `postback` data. In this method, we raise the `TextChanged` event, passing a reference to the custom control and any event arguments that are applicable, as shown in Examples 6-10 (VB) and 6-11 (C#). In this example, no arguments are required, so `EventArgs.Empty` is used for the event arguments parameter.

See Also

Recipes 6.1, 6.2, and 6.4

Example 6-7. Custom control with state (.vb)

```
Option Explicit On
Option Strict On

Imports System.Drawing

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides a custom control that maintains state through
 ''' postbacks to the server.
 ''' </summary>
 Public  Class  CH06CustomControlWithStateVB1
  Inherits WebControl
  Implements IPostBackDataHandler

  'private copies of attribute data
  Private mLabelText As String = "Label: "
  Private mTextColor As Color = Color.Black
  Private mTextboxWidth As Integer = 3

  '''**************************************************************************
  ''' <summary>
  ''' This property provides the ability to set the text of of the label
  ''' in the control
  ''' </summary>
  Public Property labelText() As String
   Get
    Return (mLabelText)
   End Get

   Set(ByVal Value As String)
    mLabelText = Value
   End Set
  End Property 'labelText


  '''**************************************************************************
  ''' <summary>
  ''' This property provides the ability to set the color of the text
  ''' in the control
  ''' </summary>
  Public Property textColor() As Color
   Get
    Return (mTextColor)
   End Get

  Set(ByVal Value As Color)
   mTextColor = Value
  End Set
 End Property 'textColor

 '''**************************************************************************
```

```
''' <summary>
''' This property provides the ability to set the width of the text box
''' in the control
''' </summary>
Public Property textboxWidth() As Integer
 Get
  Return (mTextboxWidth)
 End Get

 Set(ByVal Value As Integer)
  mTextboxWidth = Value
 End Set
End Property 'textboxWidth

'''**********************************************************************
''' <summary>
''' This property provides the ability to set the HTML tag that is used
''' as the container for the control
''' </summary>
Protected Overrides ReadOnly Property TagKey() As HtmlTextWriterTag
 Get
  Return (HtmlTextWriterTag.Div)
 End Get
End Property 'TagKey

'''**********************************************************************
''' <summary>
''' This property provides the ability to set the text in the text box
''' in the control. NOTE: The text value is stored in the ViewState
''' instead of a private variable to provide the ability to check the
''' current and previous values to determine if the text has changed.
''' </summary>
Private Const VS_TEXTBOX_VALUE As String = "TextboxValue"

Public Property text() As String
 Get
  Dim value As String = Nothing
  If (Not IsNothing(viewstate(VS_TEXTBOX_VALUE))) Then
   value = CStr(viewstate(VS_TEXTBOX_VALUE))
  End If
  Return (value)
 End Get

 Set(ByVal Value As String)
  ViewState(VS_TEXTBOX_VALUE) = Value
 End Set
End Property 'text

'''**********************************************************************
''' <summary>
''' This routine renders the HTML output of the control contents
''' </summary>
```

```
'''
''' <param name="writer">Set to the HtmlTextWriter to use to output the
''' rendered HTML for the control
''' </param>
Protected Overrides Sub RenderContents(ByVal writer As HtmlTextWriter)
 'output label within a font tag
 writer.AddAttribute("color", _
   ColorTranslator.ToHtml(textColor))
 writer.RenderBeginTag(HtmlTextWriterTag.Font)
 writer.Write(labelText)
 writer.RenderEndTag()

 'output input control
 writer.AddAttribute(HtmlTextWriterAttribute.Type, _
   "text")
 writer.AddAttribute(HtmlTextWriterAttribute.Size, _
   textboxWidth.ToString())
 'output name attribute to identify data on postback
 writer.AddAttribute(HtmlTextWriterAttribute.Name, _
   Me.UniqueID)
 'output value attribute only if value exists
 If (Not IsNothing(text)) Then
  writer.AddAttribute(HtmlTextWriterAttribute.Value, _
    text)
 End If

 writer.RenderBeginTag(HtmlTextWriterTag.Input)
 writer.RenderEndTag()
End Sub 'RenderContents

'''**********************************************************************
''' <summary>
''' This routine processes data posted back from the client
''' </summary>
'''
''' <param name="postDataKey">The key identifier for the control</param>
''' <param name="postCollection">The collection of all incoming name
''' values</param>
'''
''' <returns>set true if the server control's state changes as a result
''' of the post back; otherwise false
''' </returns>

Public Overridable Function LoadPostData(ByVal postDataKey As String, _
   ByVal postCollection As NameValueCollection) As Boolean _
   Implements IPostBackDataHandler.LoadPostData

 'set the value of the text property from the postback data
 text = postCollection(postDataKey)
 Return (False)
End Function 'LoadPostData
```

```
 '''*************************************************************************
 ''' <summary>
 ''' This routine processes data changed events as a result of the postback
 ''' </summary>
 Public Overridable Sub RaisePostDataChangedEvent() _
  Implements IPostBackDataHandler.RaisePostDataChangedEvent

 End Sub 'RaisePostDataChangedEvent
  End Class
End Namespace
```

Example 6-8. Custom control with state (.cs)

```
using System;
using System.Collections.Specialized;
using System.Drawing;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides a custom control that maintains state through
 /// postbacks to the server.
 /// </summary>
 public class CH06CustomControlWithStateCS1 :
  WebControl, IPostBackDataHandler
 {
  // private copies of attribute data
  private String mLabelText = "Label: ";
  private Color mTextColor = Color.Black;
  private int mTextboxWidth = 3;

  ///************************************************************************
  /// <summary>
  /// This property provides the ability to get/set the text of of the label
  /// in the control
  /// </summary>
  public String labelText
  {
   get
   {
    return (mLabelText);
   }
   set
   {
    mLabelText = value;
   }
  } // labelText

  ///***********************************************************************
  /// <summary>
  /// This property provides the ability to set the color of the text
  /// in the control
  /// </summary>
  public Color textColor
  {
   get
   {
    return (mTextColor);
   }
   set
   {
    mTextColor = value;
   }
  } // textColor

  ///***********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the width of the text box
  /// in the control
  /// </summary>
  public int textboxWidth
  {
   get
   {
    return (mTextboxWidth);
   }
   set
   {
    mTextboxWidth = value;
   }
  } // textboxWidth

  ///***********************************************************************
  /// <summary>
  /// This property provides the ability to set the HTML tag that is used
  /// as the container for the control
  /// </summary>
  protected override HtmlTextWriterTag TagKey
  {
   get
   {
    return (HtmlTextWriterTag.Div);
   }
  } // TagKey

  ///***********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the text in the text box
  /// in the control
  /// </summary>
  private const String VS_TEXTBOX_VALUE = "TextboxValue";

  public String text
  {
   get
   {
    String value = null;
    if (ViewState[VS_TEXTBOX_VALUE] != null)
    {
    value = (String)(ViewState[VS_TEXTBOX_VALUE]);
    }
    return (value);
   }
   set
   {
    ViewState[VS_TEXTBOX_VALUE] = value;
   }
  } // text

  ///*************************************************************************
  /// <summary>
  /// This routine renders the HTML output of the control contents
  /// </summary>
  ///
  /// <param name="writer">Set to the HtmlTextWriter to use to output the
  /// rendered HTML for the control
  /// </param>
  protected override void RenderContents(HtmlTextWriter writer)
  {
   //output label
   writer.AddAttribute("color",
     ColorTranslator.ToHtml(textColor));
   writer.RenderBeginTag(HtmlTextWriterTag.Font);
   writer.Write(labelText);
   writer.RenderEndTag();

   //output input control
   writer.AddAttribute(HtmlTextWriterAttribute.Type,
     "text");
   writer.AddAttribute(HtmlTextWriterAttribute.Size,
     textboxWidth.ToString());

   // output name attribute to identify data on postback
   writer.AddAttribute(HtmlTextWriterAttribute.Name,
     this.UniqueID);

   // output value attribute only if value exists
   if (text != null)
   {
   writer.AddAttribute(HtmlTextWriterAttribute.Value,
      text);
    }

    writer.RenderBeginTag(HtmlTextWriterTag.Input);
    writer.RenderEndTag();
  } // RenderContents

  ///***********************************************************************
  /// <summary>
  /// This routine processes data posted back from the client
  /// </summary>
  ///
  /// <param name="postDataKey">The key identifier for the control</param>
  /// <param name="postCollection">The collection of all incoming name
  /// values</param>
  ///
  /// <returns>set true if the server control's state changes as a result
  /// of the post back; otherwise false
  /// </returns>
  public virtual bool LoadPostData(string postDataKey,
          NameValueCollection postCollection)
  {
    // set the value of the text property from the postback data
    text = postCollection[postDataKey];
    return (false);
  } // LoadPostData

  ///***********************************************************************
  /// <summary>
  /// This routine processes data changed events as a result of the postback
  /// </summary>
  public virtual void RaisePostDataChangedEvent()
  {
  } // RaisePostDataChangedEvent
 } // CH06CustomControlWithStateCS1
}
```

Example 6-9. Using the custom control with state (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH06DisplayControlWithStateVB1.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH06DisplayControlWithStateVB1"
 Title="Custom Control With State" %>
<%@ Register TagPrefix="ASPCookbook" Namespace="ASPNetCookbook.VBExamples" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Custom Control With State (VB)
 </div>
 <table width="90%" align="center" border="0">
  <tr bgcolor="#ffffcc">
   <td align="center">
    <ASPCookbook:CH06CustomControlWithStateVB1
    id="ccAttributes" runat="server"
    labelText="Enter Age: "
    textColor="#000080"
    textboxWidth="5" />
   </td>
  </tr>
  <tr>
   <td align="center">
    <br/>
    <asp:Button ID="btnSubmit" runat="server"
       Text="Submit" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 6-10. Custom control with state and changed event (.vb)

```
Option Explicit On
Option Strict On

Imports System.Drawing

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides a custom control that maintains state through
 ''' postbacks to the server and raises an event when the entered
 ''' data changes.
 ''' </summary>
 Public Class CH06CustomControlWithStateVB2
  Inherits WebControl
  Implements IPostBackDataHandler
```

```
'define an event to be raised if the text changes
Public Event TextChanged As EventHandler

'private copies of attribute data
Private mLabelText As String = "Label: "
Private mTextColor As Color = Color.Black
Private mTextboxWidth As Integer = 3

'''**************************************************************************
''' <summary>
''' This property provides the ability to set the text of of the label
''' in the control
''' </summary>
Public Property labelText() As String
 Get
  Return (mLabelText)
 End Get

 Set(ByVal Value As String)
  mLabelText = Value
 End Set
End Property 'labelText

'''**************************************************************************
''' <summary>
''' This property provides the ability to set the color of the text
''' in the control
''' </summary>
Public Property textColor() As Color
 Get
  Return (mTextColor)
 End Get

 Set(ByVal Value As Color)
  mTextColor = Value
 End Set
End Property 'textColor

'''**************************************************************************
''' <summary>
''' This property provides the ability to set the width of the text box
''' in the control
''' </summary>
Public Property textboxWidth() As Integer
 Get
  Return (mTextboxWidth)
 End Get

 Set(ByVal Value As Integer)
  mTextboxWidth = Value
 End Set
End Property 'textboxWidth

'''*************************************************************************
''' <summary>
''' This property provides the ability to set the HTML tag that is used
''' as the container for the control
''' </summary>
Protected Overrides ReadOnly Property TagKey() As HtmlTextWriterTag
 Get
  Return (HtmlTextWriterTag.Div)
 End Get
End Property 'TagKey

'''*************************************************************************
''' <summary>
''' This property provides the ability to set the text in the text box
''' in the control. NOTE: The text value is stored in the ViewState
''' instead of a private variable to provide the ability to check the
''' current and previous values to determine if the text has changed.
''' </summary>
Private Const VS_TEXTBOX_VALUE As String = "TextboxValue"

Public Property text() As String
 Get
  Dim value As String = Nothing
  If (Not IsNothing(viewstate(VS_TEXTBOX_VALUE))) Then
  value = CStr(viewstate(VS_TEXTBOX_VALUE))
  End If
  Return (value)
 End Get

 Set(ByVal Value As String)
  ViewState(VS_TEXTBOX_VALUE) = Value
 End Set
End Property 'text

'''*************************************************************************
''' <summary>
''' This routine renders the HTML output of the control contents
''' </summary>
'''
''' <param name="writer">Set to the HtmlTextWriter to use to output the
''' rendered HTML for the control
''' </param>
Protected Overrides Sub RenderContents(ByVal writer As HtmlTextWriter)
 'output label within a font tag
 writer.AddAttribute("color", _
   ColorTranslator.ToHtml(textColor))
 writer.RenderBeginTag(HtmlTextWriterTag.Font)
 writer.Write(labelText)
 writer.RenderEndTag()
```

```
  'output input control
  writer.AddAttribute(HtmlTextWriterAttribute.Type, _
    "text")
  writer.AddAttribute(HtmlTextWriterAttribute.Size, _
    textboxWidth.ToString())

  'output name attribute to identify data on postback
  writer.AddAttribute(HtmlTextWriterAttribute.Name, _
    Me.UniqueID)

  'output value attribute only if value exists
  If (Not IsNothing(text)) Then
   writer.AddAttribute(HtmlTextWriterAttribute.Value, _
     text)
  End If

  writer.RenderBeginTag(HtmlTextWriterTag.Input)
  writer.RenderEndTag()
End Sub 'RenderContents

'''***************************************************************************
''' <summary>
''' This routine processes data posted back from the client
''' </summary>
'''
''' <param name="postDataKey">The key identifier for the control</param>
''' <param name="postCollection">The collection of all incoming name
''' values</param>
'''
''' <returns>set true if the server control's state changes as a result
''' of the post back; otherwise false
''' </returns>
Public Overridable Function LoadPostData(ByVal postDataKey As String,_
  ByVal postCollection As NameValueCollection) As Boolean _
  Implements IPostBackDataHandler.LoadPostData
 Dim dataChanged As Boolean = False
 Dim postbackValue As String

 'check to see if the data changed
 postbackValue = postCollection(postDataKey)
 If (Not postbackValue.Equals(text)) Then
  dataChanged = True
 End If

 'set the value of the text property from the postback data
 text = postbackValue

 Return (dataChanged)
End Function 'LoadPostData

'''***************************************************************************
''' <summary>
```

```
  ''' This routine processes data changed events as a result of the postback
  ''' </summary>
  Public Overridable Sub RaisePostDataChangedEvent() _
    Implements IPostBackDataHandler.RaisePostDataChangedEvent
   RaiseEvent TextChanged(Me, EventArgs.Empty)
  End Sub 'RaisePostDataChangedEvent
 End Class 'CH06CustomControlWithStateVB2
End Namespace
```

Example 6-11. Custom control with state and changed event (.cs)

```
using System;
using System.Collections.Specialized;
using System.Drawing;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides a custom control that maintains state through
 /// postbacks to the server and raises an event when the entered
 /// data changes.
 /// </summary>
 public class CH06CustomControlWithStateCS2 :
  WebControl, IPostBackDataHandler
 {
  // define an event to be raised if the text changes
  public event EventHandler TextChanged;

  // private copies of attribute data
  private String mLabelText = "Label: ";
  private Color mTextColor = Color.Black;
  private int mTextboxWidth = 3;

  ///*********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the text of of the label
  /// in the control
  /// </summary>
  public String labelText
  {
   get
   {
    return (mLabelText);
   }
   set
  {
   mLabelText = value;
  }
} // labelText

///***********************************************************************
/// <summary>
/// This property provides the ability to set the color of the text
/// in the control
/// </summary>
public Color textColor
{
  get
  {
   return (mTextColor);
  }
  set
  {
   mTextColor = value;
  }
} // textColor

///***********************************************************************
/// <summary>
/// This property provides the ability to get/set the width of the text box
/// in the control
/// </summary>
public int textboxWidth
{
  get
  {
   return (mTextboxWidth);
  }
  set
  {
   mTextboxWidth = value;
  }
} // textboxWidth

///***********************************************************************
/// <summary>
/// This property provides the ability to set the HTML tag that is used
/// as the container for the control
/// </summary>
protected override HtmlTextWriterTag TagKey
{
  get
  {
   return (HtmlTextWriterTag.Div);
  }
} // TagKey
```

```
 ///*************************************************************************
/// <summary>
/// This property provides the ability to get/set the text in the text box
/// in the control
/// </summary>
private const String VS_TEXTBOX_VALUE = "TextboxValue";

public String text
{
 get
 {
  String value = null;
  if (ViewState[VS_TEXTBOX_VALUE] != null)
  {
  value = (String)(ViewState[VS_TEXTBOX_VALUE]);
  }
  return (value);
 }
 set
 {
  ViewState[VS_TEXTBOX_VALUE] = value;
 }
} // text

 ///*************************************************************************
/// <summary>
/// This routine renders the HTML output of the control contents
/// </summary>
///
/// <param name="writer">Set to the HtmlTextWriter to use to output the
/// rendered HTML for the control
/// </param>
protected override void RenderContents(HtmlTextWriter writer)
{
 //output label
 writer.AddAttribute("color",
   ColorTranslator.ToHtml(textColor));
 writer.RenderBeginTag(HtmlTextWriterTag.Font);
 writer.Write(labelText);
 writer.RenderEndTag();

 //output input control
 writer.AddAttribute(HtmlTextWriterAttribute.Type,
   "text");
 writer.AddAttribute(HtmlTextWriterAttribute.Size,
   textboxWidth.ToString());

 // output name attribute to identify data on postback
 writer.AddAttribute(HtmlTextWriterAttribute.Name,
   this.UniqueID);

 // output value attribute only if value exists
 if (text != null)
 {
  writer.AddAttribute(HtmlTextWriterAttribute.Value,
    text);
 }

 writer.RenderBeginTag(HtmlTextWriterTag.Input);
 writer.RenderEndTag();
} // RenderContents

///***************************************************************************
/// <summary>
/// This routine processes data posted back from the client
/// </summary>
///
/// <param name="postDataKey">The key identifier for the control</param>
/// <param name="postCollection">The collection of all incoming name
/// values</param>
///
/// <returns>set true if the server control's state changes as a result
/// of the post back; otherwise false
/// </returns>

public virtual bool LoadPostData(string postDataKey,
     NameValueCollection postCollection)
{
 Boolean dataChanged = false;
 String postbackValue;

 // check to see if the data changed
 postbackValue = postCollection[postDataKey];
 if (!postbackValue.Equals(text))
 {
  dataChanged = true;
 }

 // set the value of the text property from the postback data
 text = postbackValue;

 return (dataChanged);
} // LoadPostData

///***************************************************************************
/// <summary>
/// This routine processes data changed events as a result of the postback
/// </summary>
public virtual void RaisePostDataChangedEvent()
{
 // raise event if a handler is assigned
 if (TextChanged != null)
 {
  TextChanged(this, EventArgs.Empty);
```

```
    }
   } // RaisePostDataChangedEvent
 } //  CH06CustomControlWithStateCS2
}
```

Example 6-12. Using the custom control with state and changed event (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH06DisplayControlWithStateVB2.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH06DisplayControlWithStateVB2"
 Title="Custom Control With State" %>
<%@ Register TagPrefix="ASPCookbook" Namespace="ASPNetCookbook.VBExamples" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Custom Control With State And Events (VB)
 </div>
 <table width="90%" align="center" border="0">
  <tr bgcolor="#ffffcc">
   <td align="center">
    <ASPCookbook:CH06CustomControlWithStateVB2
    id="ccAttributes" runat="server"
    labelText="Enter Age: "
    textColor="#000080"
    textboxWidth="5"
    OnTextChanged="ccAttributes_TextChanged" />
   </td>
  </tr>
  <tr>
   <td align="center">
    <asp:Label ID="labMessage" Runat="server" />
   </td>
  </tr>
  <tr>
   <td align="center">
    <br/>
    <asp:Button ID="btnSubmit" runat="server"
           Text="Submit" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 6-13. Using the custom control with state and changed event code-behind (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code beside for
   '''  CH06DisplayControlWithStateVB2.aspx
  ''' </summary>
   Partial  Class  CH06DisplayControlWithStateVB2
     Inherits System.Web.UI.Page

 '''**************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   labMessage.Text = ""
 End Sub 'Page_Load

 '''**************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the custom control text
 ''' changed event.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub ccAttributes_TextChanged(ByVal sender As Object, _
           ByVal e As System.EventArgs)
  labMessage.Text = "Data Changed"
  End Sub 'ccAttributes_TextChanged
  End  Class  'CH06DisplayControlWithStateVB2
End Namespace
```

Example 6-14. Using the custom control with state and changed event code-behind (.cs)

```
using System;
namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code beside for
   ///  CH06DisplayControlWithStateCS2.aspx
  /// </summary>
   public partial class CH06DisplayControlWithStateCS2 : System.Web.UI.Page
  {
    ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
   labMessage.Text = "";
 } // Page_Load

 ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the custom control text
 /// changed event.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void ccAttributes_TextChanged(Object sender,
             System.EventArgs e)
 {
  labMessage.Text = "Data Changed";
 } //ccAttributes_TextChanged
  } //  CH06DisplayControlWithStateCS2
}
```

PREV

Recipe 6.5. Using the Control State with Custom Controls

Problem

You want to allow users to disable the `ViewState` for your custom control but you do not want to lose functionality when they do. In other words, when a developer disables the `ViewState` for the page or your custom control, you want to be able to maintain critical state information for your control.

Solution

Create a custom control like the one described in Recipe 6.3, register the control with the `Page` indicating the control requires use of the `Control State`, override the `SaveControlState` method to save the critical information for your control in the `Control State`, and override the `LoadControlState` method to reload the critical information from the `Control State` on `postback`.

Use the .NET language of your choice to:

1. Create a class that inherits from the `WebControl` class in the `System.Web.UI.WebControls` namespace.
2. Implement state support as described in Recipe 6.3.
3. In the control `Init` method, register the control with the `Page` indicating the control requires use of the `Control State`.
4. Override the `SaveControlState` method to save the critical information for your control in the `Control State`.
5. Override the `LoadControlState` method to reload the control information from the `Control State` on `postback`.

To use the custom control in an ASP.NET page:

1. Register the assembly containing the control.
2. Insert the tag for the custom control anywhere in the page and set the attributes appropriately.

Examples 6-15 and 6-16 show the VB and C# class files for a custom control that maintains its critical state information in the Control State. Examples 6-17, Examples 6-18 through 6-19 show how to use the custom control in an ASP.NET page.

Discussion

Custom controls developed in ASP.NET 1.x had only one way to save critical state information. The only option available was to use the `ViewState`, which worked well until a developer disabled the `ViewState` for the page or your custom control. When he disabled the `ViewState`, the state information needed by your control was unavailable at `postback`, generally resulting in loss of functionality for your control.

ASP.NET 2.0 provides another method of saving the critical state information. This method uses the `Control State` to save the state information in the page in the same manner as the `ViewState`, but the `Control State` cannot be disabled. Saving the critical state information in the `Control State` provides the ability for the developer to improve performance by disabling the `ViewState` without compromising the functionality of your control.

The `Control State` should be used only for small amounts of critical state information that your control must have upon `postback`. The information you place in the `Control State` is stored in the rendered page as a hidden input control (as part of same hidden control that stores the `ViewState`). This results in the information being sent to the browser with the page request as well as being sent back to the server upon `postback`, which will affect performance if significant amounts of data are stored in the `Control State`.

To store state information in the `Control State`, you will need to implement a control similar to the one described in Recipe 6.3. One modification is required to the control from what is described in Recipe 6.3. You need to remove any property storage from the `ViewState`, as shown below for the text property:

```
Public Property text() As String
 Get
  Return (mText)
 End Get

 Set(ByVal Value As String)
  mText = Value
 End Set
End Property 'text


public String text
{
  get
  {
    return (mText);
  }
  set
  {
 mText = value;
```

```
    }
  } // text
```

Next, you need to register the control with the page to inform the page that the Control State is required for your control. The registration is normally performed in the Init event handler for the control.

```
     Private Sub CH06CustomControlWithStateVB1_Init(ByVal sender As Object, _
            ByVal e As EventArgs) _
         Handles Me.Init
   Page.RegisterRequiresControlState(Me)
  End Sub 'CH06CustomControlWithStateVB1_Init
```

```
  protected override void OnInit(EventArgs e)
  {
    base.OnInit(e);
    Page.RegisterRequiresControlState(this);
  }
```

Then you need to override the SaveControlState method to save the critical state information for your control in the Control State. Storing data in the Control State requires verifying that you have data to store and checking to see if other data is being stored in the Control State, as shown in Examples 6-15 (VB) and 6-16 (C#).

All data stored in the Control State is managed by the Page and is stored as object Pairs. When storing your data in the Control State, you must check for data stored in the Control State and add your data accordingly. Failure to handle previously added data or incorrectly adding your data will affect the operation of the page and other controls on the page.

Finally, you need to override the LoadControlState method to reload your critical state information upon postback. This requires checking to see if any data is available and, if so, evaluating its type. If the data type is a Pair, you must pass the First property of the pair to the base class and use the Second property as your data. If the data type is not a Pair, you will need to verify the data type is the type you are expecting for your control, as shown in Examples 6-15 (VB) and 6-16 (C#).

When the state data provided to the LoadControlState method is of type Pair, you must pass the data to the base class. Failure to do so will result in incorrect operation of the page or other controls on the page.

The Control State is a valuable asset for custom control development. With it, you can ensure

critical state information is available while providing the developer the ability to enhance performanc
by disabling the `ViewState`.

See Also

Recipes 6.3 and 19.1

Example 6-15. Custom control using the control state (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Drawing

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides a custom control that maintains state through
  ''' postbacks to the server using the page control state.
  ''' </summary>
  Public Class CH06CustomControlWithStateVB3
    Inherits WebControl
 Implements IPostBackDataHandler

 'define an event to be raised if the text changes
 Public Event TextChanged As EventHandler

 'private copies of attribute data
 Private mLabelText As String = "Label: "
 Private mTextColor As Color = Color.Black
 Private mTextboxWidth As Integer = 3
 Private mText As String = Nothing

 '''**************************************************************************
 ''' <summary>
 ''' This property provides the ability to set the text of of the label
 ''' in the control
 ''' </summary>
 Public Property labelText( ) As String
   Get
  Return (mLabelText)
   End Get

   Set(ByVal Value As String)
  mLabelText = Value
   End Set
 End Property 'labelText
```

```
'''**************************************************************************
''' <summary>
''' This property provides the ability to set the color of the text
''' in the control
''' </summary>
Public Property textColor( ) As Color
  Get
 Return (mTextColor)
  End Get
  Set(ByVal Value As Color)
 mTextColor = Value
  End Set
End Property 'textColor

'''**************************************************************************
''' <summary>
''' This property provides the ability to set the width of the text box
''' in the control
''' </summary>
Public Property textboxWidth( ) As Integer
 Get
  Return (mTextboxWidth)
 End Get

 Set(ByVal Value As Integer)
  mTextboxWidth = Value
 End Set
End Property 'textboxWidth

'''**************************************************************************
''' <summary>
''' This property provides the ability to set the HTML tag that is used
''' as the container for the control
''' </summary>
Protected Overrides ReadOnly Property TagKey( ) As HtmlTextWriterTag
 Get
  Return (HtmlTextWriterTag.Div)
 End Get
End Property 'TagKey

'''**************************************************************************
''' <summary>
''' This property provides the ability to get/set the text in the text box
''' in the control.
''' </summary>
Public Property text( ) As String
 Get
  Return (mText)
 End Get

 Set(ByVal Value As String)
  mText = Value
 End Set
End Property 'text

'''***************************************************************************
''' <summary>
''' This routine renders the HTML output of the control contents
''' </summary>
'''
''' <param name="writer">Set to the HtmlTextWriter to use to output the
''' rendered HTML for the control
''' </param>
Protected Overrides Sub RenderContents(ByVal writer As HtmlTextWriter)
  'output label within a font tag
  writer.AddAttribute("color", _
      ColorTranslator.ToHtml(textColor))
  writer.RenderBeginTag(HtmlTextWriterTag.Font)
  writer.Write(labelText)
  writer.RenderEndTag( )

  'output input control
  writer.AddAttribute(HtmlTextWriterAttribute.Type, _
     "text")
  writer.AddAttribute(HtmlTextWriterAttribute.Size, _
     textboxWidth.ToString( ))

  'output name attribute to identify data on postback
  writer.AddAttribute(HtmlTextWriterAttribute.Name, _
     Me.UniqueID)

  'output value attribute only if value exists
  If (Not IsNothing(text)) Then
 writer.AddAttribute(HtmlTextWriterAttribute.Value, _
   text)
  End If

  writer.RenderBeginTag(HtmlTextWriterTag.Input)
  writer.RenderEndTag( )
End Sub 'RenderContents

'''***************************************************************************
''' <summary>
''' This routine processes data posted back from the client
''' </summary>
'''
''' <param name="postDataKey">The key identifier for the control</param>
''' <param name="postCollection">The collection of all incoming name
''' values</param>
'''
''' <returns>set true if the server control's state changes as a result
''' of the post back; otherwise false
''' </returns>
Public Overridable Function LoadPostData(ByVal postDataKey As String, _
   ByVal postCollection As NameValueCollection) As Boolean _
    Implements IPostBackDataHandler.LoadPostData
  Dim dataChanged As Boolean = False
  Dim postbackValue As String

  'check to see if the data changed
  postbackValue = postCollection(postDataKey)
  If (Not postbackValue.Equals(text)) Then
    dataChanged = True
  End If

  'set the value of the text property from the postback data
  text = postbackValue

  Return (dataChanged)
End Function 'LoadPostData

'''*************************************************************************
''' <summary>
''' This routine processes data changed events as a result of the postback
''' </summary>
Public Overridable Sub RaisePostDataChangedEvent( ) _
    Implements IPostBackDataHandler.RaisePostDataChangedEvent
   RaiseEvent TextChanged(Me, EventArgs.Empty)
End Sub 'RaisePostDataChangedEvent

'''*************************************************************************
''' <summary>
''' This routine provides the event handler for the control init event.
''' It is responsible for registering with the page to indicate this
''' control requires the control state
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Private Sub CH06CustomControlWithStateVB1_Init(ByVal sender As Object, _
             ByVal e As EventArgs) _
  Handles Me.Init
  Page.RegisterRequiresControlState(Me)
End Sub 'CH06CustomControlWithStateVB1_Init

'''*************************************************************************
''' <summary>
''' This routine provides the ability to restore the data from the control
''' state that was previously saved by the SaveControlState method
''' </summary>
'''
''' <param name="state">Set to the control state to restore</param>
Protected Overrides Sub LoadControlState(ByVal state As Object)
   Dim statePair As Pair
'check to see if there is any control state to restore
If (Not state Is Nothing) Then
```

```
 'check to see if the state information contain base data
    If (TypeOf state Is Pair) Then
    'state contains control state information for the base class so
    'extract it and pass it along to the base class
    statePair = CType(state, Pair)
    MyBase.LoadControlState(statePair.First)

    'get the state information for this object
    text = CStr(statePair.Second)
 Else
  'state contains only simple data so check to see if it is
  'applicable to this object
  If (TypeOf state Is String) Then
   'get the state information for this object
   text = CStr(state)
  Else
   'pass state information on to base object
   MyBase.LoadControlState(state)
  End If
  End If
      End If
End Sub 'LoadControlState

'''***************************************************************************
''' <summary>
''' This routine provides the ability to save information in the control
''' state
''' </summary>
'''
''' <returns>An object containing the information to store in the control
''' state
''' </returns>
Protected Overrides Function SaveControlState( ) As Object
 Dim baseControlState As Object
 Dim returnValue As Object = Nothing

 baseControlState = MyBase.SaveControlState( )
 If (text Is Nothing) Then
  returnValue = baseControlState
 Else
  If (baseControlState Is Nothing) Then
   returnValue = text
  Else
   returnValue = New Pair(baseControlState, _
      text)
  End If
 End If

 Return (returnValue)
End Function 'SaveControlState
End  Class  'CH06CustomControlWithStateVB3
 End Namespace
```

Example 6-16. Custom control using the control state (.cs)

```
using System;
using System.Collections.Specialized;
using System.Drawing;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides a custom control that maintains state through
  /// postbacks to the server and raises an event when the entered
  /// data changes.
  /// </summary>
   public class CH06CustomControlWithStateCS3 :
     WebControl, IPostBackDataHandler
  {
 // define an event to be raised if the text changes
 public event EventHandler TextChanged;

 // private copies of attribute data
 private String mLabelText = "Label: ";
 private Color mTextColor = Color.Black;
 private int mTextboxWidth = 3;
 private String mText = null;

 ///*************************************************************************
 /// <summary>
 /// This property provides the ability to get/set the text of of the label
 /// in the control
 /// </summary>
 public String labelText
 {
  get
  {
   return (mLabelText);
  }
  set
  {
   mLabelText = value;
  }
 } // labelText

 ///*************************************************************************
 /// <summary>
 /// This property provides the ability to set the color of the text
 /// in the control
 /// </summary>
 public Color textColor
 {
  get
  {
   return (mTextColor);
  }
  set
  {
   mTextColor = value;
  }
 } // textColor

 ///***************************************************************************
 /// <summary>
 /// This property provides the ability to get/set the width of the text box
 /// in the control
 /// </summary>
 public int textboxWidth
 {
  get
  {
   return (mTextboxWidth);
  }
  set
  {
   mTextboxWidth = value;
  }
 } // textboxWidth

 ///***************************************************************************
 /// <summary>
 /// This property provides the ability to set the HTML tag that is used
 /// as the container for the control
 /// </summary>
 protected override HtmlTextWriterTag TagKey
 {
  get
  {
   return (HtmlTextWriterTag.Div);
  }
 } // TagKey

 ///***************************************************************************
 /// <summary>
 /// This property provides the ability to get/set the text in the text box
 /// in the control
 /// </summary>
 public String text
 {
  get
  {
   return (mText);
  }
  set
  {
   mText = value;
  }
 } // text

 ///*************************************************************************
 /// <summary>
 /// This routine renders the HTML output of the control contents
 /// </summary>
 ///
 /// <param name="writer">Set to the HtmlTextWriter to use to output the
 /// rendered HTML for the control
 /// </param>
 protected override void RenderContents(HtmlTextWriter writer)
 {
  //output label
  writer.AddAttribute("color",
    ColorTranslator.ToHtml(textColor));
  writer.RenderBeginTag(HtmlTextWriterTag.Font);
  writer.Write(labelText);
  writer.RenderEndTag( );

  //output input control
  writer.AddAttribute(HtmlTextWriterAttribute.Type,
    "text");
  writer.AddAttribute(HtmlTextWriterAttribute.Size,
    textboxWidth.ToString( ));

  // output name attribute to identify data on postback
  writer.AddAttribute(HtmlTextWriterAttribute.Name,
    this.UniqueID);

  // output value attribute only if value exists
  if (text != null)
  {
   writer.AddAttribute(HtmlTextWriterAttribute.Value,
     text);
  }

  writer.RenderBeginTag(HtmlTextWriterTag.Input);
  writer.RenderEndTag( );
 } // RenderContents
 ///*************************************************************************
 /// <summary>
 /// This routine processes data posted back from the client
 /// </summary>
 ///
 /// <param name="postDataKey">The key identifier for the control</param>
 /// <param name="postCollection">The collection of all incoming name
 /// values</param>
 ///
 /// <returns>set true if the server control's state changes as a result
 /// of the post back; otherwise false
 /// </returns>
 public virtual bool LoadPostData(string postDataKey,
       NameValueCollection postCollection)
 {
  Boolean dataChanged = false;
  String postbackValue;

  // check to see if the data changed
  postbackValue = postCollection[postDataKey];
  if (!postbackValue.Equals(text))
  {
   dataChanged = true;
  }

  // set the value of the text property from the postback data
  text = postbackValue;

  return (dataChanged);
 } // LoadPostData

 ///*************************************************************************
 /// <summary>
 /// This routine processes data changed events as a result of the postback
 /// </summary>
 public virtual void RaisePostDataChangedEvent( )
 {
  // raise event if a handler is assigned
  if (TextChanged != null)
  {
   TextChanged(this, EventArgs.Empty);
  }
 } // RaisePostDataChangedEvent

 ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the control init event.
 /// It is responsible for registering with the page to indicate this
 /// control requires the control  state6
 /// </summary>
 ///
 /// <param name="e">Set to the event arguments</param>
 protected override void OnInit(EventArgs e)
 {
  base.OnInit(e);
  Page.RegisterRequiresControlState(this);
 }
```

```
///***********************************************************************
/// <summary>
/// This routine provides the ability to restore the data from the control
/// state that was previously saved by the SaveControlState method
/// </summary>
///
/// <param name="state">Set to the control state to restore</param>
protected override void LoadControlState(Object state)
{
 Pair statePair;

 // check to see if there is any control state to restore
 if (state != null)
 {
  // check to see if the state information contain base data
  if (state is Pair)
  {
   // state contains control state information for the base class so
   // extract it and pass it along to the base class
   statePair = (Pair)state;
   base.LoadControlState(statePair.First);
   // get the state information for this object
   text = (String)(statePair.Second);
  }
  else
  {
   // state contains only simple data so check to see if it is
   // applicable to this object
   if (state is String)
   {
   // get the state information for this object
   text = (String)state;
   }
   else
   {
   // pass state information on to base object
   base.LoadControlState(state);
   }
  }
 }
} // LoadControlState

///***********************************************************************
/// <summary>
/// This routine provides the ability to save information in the control
/// state
/// </summary>
///
/// <returns>An object containing the information to store in the control
/// state
/// </returns>
protected override Object SaveControlState( )
```

```
 {
  Object baseControlState;
  Object returnValue = null;

  baseControlState = base.SaveControlState( );
  if (text == null)
  {
   returnValue = baseControlState;
  }
  else
  {
   if (baseControlState == null)
   {
    returnValue = text;
   }
   else
   {
    returnValue = new Pair(baseControlState,
       text);
   }
  }
  return (returnValue);
 } // SaveControlState
 } //  CH06CustomControlWithStateCS3
}
```

Example 6-17. Using the custom control using the control state (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH06DisplayControlWithStateVB3.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH06DisplayControlWithStateVB3"
  Title="Custom Control Using Control State" %>
<%@ Register TagPrefix="ASPCookbook" Namespace="ASPNetCookbook.VBExamples" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">
    Custom Control With State & Events Using Control State (VB)
  </div>
  <table width="90%" align="center" border="0">
    <tr bgcolor="#ffffcc">
   <td align="center">
     <ASPCookbook:CH06CustomControlWithStateVB3
    id="ccAttributes" runat="server"
    labelText="Enter Age: "
    textColor="#000080"
    textboxWidth="5"
    OnTextChanged="ccAttributes_TextChanged"
    EnableViewState="false" />
   </td>
 </tr>
 <tr>
  <td align="center">
    <asp:Label ID="labMessage" Runat="server" />
  </td>
 </tr>
 <tr>
  <td align="center">
    <br/>
    <asp:Button ID="btnSubmit" runat="server"
     Text="Submit" />
  </td>
 </tr>
  </table>
</asp:Content>
```

Example 6-18. Using the custom control using the control state code-behind (.vb)

```
Option Explicit On
Option Strict On
Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code beside for
  '''  CH06DisplayControlWithStateVB3.aspx
  ''' </summary>
  Partial  Class  CH06DisplayControlWithStateVB3
    Inherits System.Web.UI.Page

 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
    labMessage.Text = ""
 End Sub 'Page_Load

 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the custom control text
 ''' changed event.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub ccAttributes_TextChanged(ByVal sender As Object, _
          ByVal e As System.EventArgs)
  labMessage.Text = "Data Changed"
 End Sub 'ccAttributes_TextChanged
  End Class 'CH06DisplayControlWithStateVB3
End Namespace
```

Example 6-19. Using the custom control using the control state code-behind (.cs)

```
using System;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code beside for
 ///  CH06DisplayControlWithStateCS3.aspx
 /// </summary>
 public partial class CH06DisplayControlWithStateCS3 : System.Web.UI.Page
 {
  ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
     labMessage.Text = "";
  } // Page_Load

  ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the custom control text
  /// changed event.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void ccAttributes_TextChanged(Object sender,
         System.EventArgs e)
```

```
   {
    labMessage.Text = "Data Changed";
   } //ccAttributes_TextChanged
  }  //  CH06DisplayControlWithStateCS3
 }
```

PREV

PREV

Recipe 6.6. Customizing an ASP.NET TextBox Server Control

Problem

You want to customize an ASP.NET `TextBox` server control to allow only numeric input.

Solution

Create a custom control that inherits from the ASP.NET text box control and add code to emit client-side script that limits the input to only numeric values.

Use the .NET language of your choice to:

1. Create a class that inherits from the `TextBox` class in the `System.Web.UI.WebControls` namespace.
2. Override the `OnPreRender` method to have it generate the requisite client-side script.
3. Override the `AddAttributesToRender` if you need to add an attribute to the rendered control.

To use the custom control in an ASP.NET page:

1. Register the assembly containing the control.
2. Insert the tag for the custom control anywhere in the page.

Examples 6-20 and 6-21 show the VB and C# class files for an example custom control we have written to illustrate our approach. This custom control emits client-side script that checks key presse and allows only numeric keys to be entered into a text box.Example 6-22 shows how to use the custom control in an ASP.NET page.

Discussion

Extending an existing ASP.NET server control is an easy way to create the functionality you need for an application. By inheriting your custom controls from existing controls, you only have to write the code you need to add your special functionality.

To illustrate this approach, we've implemented a text box control that allows onlynumeric input, a common project requirement. Why is it necessary to implement a custom control to accomplish this? First, none of the ASP.NET controls provides this functionality. Second, although you can check the data entered in a text box to ensure it is numeric and within a range with a range`validator`, this does not prevent the user from entering letters into the text box in the first place. Extending the standard text box control by adding client-side script that allows only numeric keys to be processed and their values entered into the text box is a better solution.

The first step in extending the ASP.NET text box control is to create a custom control class that inherits from `System.Web.UI.WebControls.TextBox`.

Next, you must override the `OnPreRender` method to generate the client-side script. (The reason for overriding `OnPreRender` is that you need to get the script onto the page before the control is rendered.) This is done by creating a string containing the script block, and then registering it to be output to the page with the `Page.ClientScript`. `RegisterClientScriptBlock` method.

The client script must be created and output in an event that occurs before the `Render` event, or the script will not be output in the rendered page. This can be done in the `Load` or `PreRender` events.

Be sure to call the base class `OnPreRender` method. Failure to do so can result in lost functionality if the base control performs any operations in the`OnPreRender` method.

Output the client script using the `Page.ClientScript.RegisterClientScriptBlock` method with the first parameter set to the type of your control. This ensures that client script is output to the page only once if multiple instances of your control are used on a page. If the client script was output directly in the`Render` event and the page contained multiple instances of the custom control, the client script would be output for each instance of the control.

The client-side script output to the page in our example is shown next. It checks each key press to see if the `keycode` is in the 09 range and returns`TRue` if it is. Otherwise, it returns `false`.

```
<script type="text/javascript">
<!-
 function checkKey()
 {
  var key = event.keyCode;
  var processKey = false;
  if ((key >= 0x30) && (key <= 0x39))
  {
   processKey = true;
  }
```

```
    return(processKey);
  }
// -->
</script>
```

This script will work only for Internet Explorer. You can use the `Page.Request.Browser` object in the `OnPreRender` method to determine the browser type and version so you can output code specific to the browser requesting the page.

The last step to implement our sample custom control is to add an attribute to the renderedinput control to cause the `checkKey` method to be executed when a key is pressed with the focus set to the input control. This is done by overriding the`AddAttributesToRender` method and adding the `onkeypress` attribute.

If your application has a similar requirement to add an attribute to the rendered input control, call the base class`AddAttributesToRender` method. Failure to do so can result in lost functionality if the base control performs any operations in the `AddAttributesToRender` method.

The rendered HTML input control is shown here:

```
<input  name="ctl00$PageBody$ccNumericInput"  type="text"
     maxlength="10" size="10"
      id="ctl00_PageBody_ccNumericInput"
    onkeypress="return checkKey();" />
```

To complete the example, we insert the control into an ASP.NET page by first registering the control (see Example 6-22). We then place the control tag in the page just as if it were a plain-vanilla text box.

One of the great advantages of inheriting from an existing ASP.NET server control is the support for all the control's inherent functionality. In this case, we can set any of the attributes available for the `asp:Textbox` control, though we did no coding for those properties in our control:

```
<ASPCookbook:CH06CustomControlNumericInputVB
   id="ccNumericInput" runat="server"
   Columns="10" MaxLength="10" />
```

See Also

JavaScript: The Definitive Guide, by David Flanagan (O'Reilly), for more information on browser-specific JavaScript; *ASP.NET in a Nutshell*, by G. Andrew Duthie and Matthew MacDonald (O'Reilly); and the MSDN Library for more on `OnPreRender` and `AddAttributestoRender`

Example 6-20. Numeric input-only text box (.vb)

```
Option Explicit On
Option Strict On

Imports System.Environment
Imports System.Web.UI

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides a custom control that implements an input control
  ''' that only allows numbers to be input.
  ''' </summary>
 Public  Class  CH06CustomControlNumericInputVB
   Inherits System.Web.UI.WebControls.TextBox

   'the following constant defines the name of the name of the client-side
   'JavaScript method used to process keystrokes
   Private Const CHECK_KEY_NAME As String = "checkKey"

 '''**************************************************************************
 ''' <summary>
 ''' This routine handles the prerender event for the custom control.
 ''' It adds clientside script to process keys before adding them to
 ''' the text box.
 ''' </summary>
 '''
 ''' <param name="e">Set to the event arguments</param>
 Protected Overrides Sub OnPreRender(ByVal e As System.EventArgs)
   Dim scriptBlock As StringBuilder

     MyBase.OnPreRender(e)

  'generate code to check the key pressed
  scriptBlock = New StringBuilder
  scriptBlock.Append("function " & CHECK_KEY_NAME & "( )" & NewLine)
  scriptBlock.Append("{" & NewLine)
  scriptBlock.Append("var key = event.keyCode;" & NewLine)
  scriptBlock.Append("var processKey = false;" & NewLine)
  scriptBlock.Append("if ((key >= 0x30) && (key <= 0x39))" & NewLine)
  scriptBlock.Append("{" & NewLine)
  scriptBlock.Append("processKey = true;" & NewLine)
  scriptBlock.Append("}" & NewLine)
  scriptBlock.Append("return(processKey);" & NewLine)
  scriptBlock.Append("}" & NewLine)

  'register script to be output when the page is rendered
  Page.ClientScript.RegisterStartupScript(Me.GetType( ), _
         CHECK_KEY_NAME, _
         scriptBlock.ToString( ), _
         True)
 End Sub 'OnPreRender

 '''*********************************************************************
 ''' <summary>
 ''' This routine handles the AddAttributeToRender event for the custom
 ''' control. It adds the onkeypress attribute to the text box to cause
 ''' processing of all keys pressed when the text box has focus.
 ''' </summary>
 '''
 ''' <param name="writer">Set to the HtmlTextWriter to use to output the
 ''' rendered HTML for the control
 ''' </param>
 Protected Overrides Sub AddAttributesToRender( _
       ByVal writer As HtmlTextWriter)
  MyBase.AddAttributesToRender(writer)
  'add an attribute to the text box to call client script to check
  'keys pressed
  writer.AddAttribute("onkeypress", _
  "return " & CHECK_KEY_NAME & "( );")
 End Sub 'AddAttributesToRender
  End  Class  'CH06CustomControlNumericInputVB
End Namespace
```

Example 6-21. Numeric input-only text box (.cs)

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

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides a custom control that implements an input control
 /// that only allows numbers to be input.
 /// </summary>
 public class CH06CustomControlNumericInputCS : TextBox
 {
  // the following constant defines the name of the name of the client-side
  // JavaScript method used to process keystrokes
  private const String CHECK_KEY_NAME = "checkKey";

  ///***********************************************************************
  /// <summary>
  /// This routine handles the prerender event for the custom control. It
  /// adds clientside script to process keys before adding them to the
  /// text box.
  /// </summary>
  ///
  /// <param name="e">Set to the event arguments</param>
  protected override void OnPreRender(System.EventArgs e)
  {
    StringBuilder scriptBlock = null;

    base.OnPreRender(e);

    // generate code to check the key pressed
    scriptBlock = new StringBuilder( );
    scriptBlock.Append("function " + CHECK_KEY_NAME + "( )\r");
    scriptBlock.Append("{\r");
    scriptBlock.Append("var key = event.keyCode;\r");
    scriptBlock.Append("var processKey = false;\r");
    scriptBlock.Append("if ((key >= 0x30) && (key <= 0x39))\r");
    scriptBlock.Append("{\r");
    scriptBlock.Append("processKey = true;\r");
    scriptBlock.Append("}\r");
    scriptBlock.Append("return(processKey);\r");
    scriptBlock.Append("}\r");
    // register script to be output when the page is rendered
    Page.ClientScript.RegisterStartupScript(this.GetType( ),
         CHECK_KEY_NAME,
         scriptBlock.ToString( ),
         true);
 } // OnPreRender

 //************************************************************************
 //
 // ROUTINE: AddAttributesToRender
 //
 // DESCRIPTION:
 //------------------------------------------------------------------------
 /// <summary>
 /// This routine handles the AddAttributeToRender event for the custom
 /// control. It adds the onkeypress attribute to the text box to cause
 /// processing of all keys pressed when the text box has focus.
 /// </summary>
 ///
 /// <param name="writer">Set to the HtmlTextWriter to use to output the
 /// rendered HTML for the control
 /// </param>
 protected override void AddAttributesToRender(
       System.Web.UI.HtmlTextWriter writer)
 {
  base.AddAttributesToRender(writer);

  // add an attribute to the text box to call client script to check
  // keys pressed
  writer.AddAttribute("onkeypress", "return " + CHECK_KEY_NAME + "( );");
 } // AddAttributesToRender
  } // CH06CustomControlNumericInputCS
}
```

Example 6-22. Using the numeric input custom control (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH06DisplayControlWithNumericInputVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH06DisplayControlWithNumericInputVB"
  Title="Custom Control With Numeric Input" %>
<%@ Register TagPrefix="ASPCookbook" Namespace="ASPNetCookbook.VBExamples" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">
 Custom Numeric Input Control (VB)
  </div>
  <table width="90%" align="center" border="0">
    <tr bgcolor="#ffffcc">
   <td align="center">
     <ASPCookbook:CH06CustomControlNumericInputVB
   id="ccNumericInput" runat="server"
   Columns="10" MaxLength="10" />
  </td>
   </tr>
   <tr>
  <td align="center">
   <br/>
   <asp:Button ID="btnSubmit" runat="server"
    Text="Submit" />
  </td>
   </tr>
   </table>
</asp:Content>
```

PREV

Chapter 7. Maintaining State

PREV

PREV

7.0 Introduction

Because HTTP is an inherently stateless protocol, you must use special techniques when you want to preserve information about users as they move from one page to the next or when they leave and reenter your application. Saving this information is known as*saving* or *maintaining state*. You need to maintain state to improve the user's experience with an ASP.NET application. Bymaintaining state, you can maintain the continuity between pages and between sessions that users demand of a webbased application, such as keeping track of items in a shopping cart or noting viewing preferences. You can enhance the performance of heavily used applications by making commonly used data available to any user, without making repeated trips to a database.

Be aware, however, that maintaining state can sometimes result in a decline in performance. For example, if you place large objects in session state you can negatively affect an application's performance by tying up system resources.

You can preserve information at the application, session, and page levels of an ASP.NET application. The recipes in this chapter demonstrate how each is done:

Application state

By making commonly used data available to all users of an application, you can sometimes improve application performance. Recipe 7.1 shows how to retrieve data from a database, place it in the `Application` object, and make it accessible to all users of an application. This is known as maintaining state at the application level.

Session state

Experienced web users expect you to remember who they are for the duration of their sessions at your site. Preserving this information, as well as information about their activities, is known as saving *session state*. Recipe 7.2 shows how to use an object to provide a container for some simple personalized data that is used by many pages of an application. The advantage is you can maintain information for each user without having to access the database each time the data is needed.

Page state

Saving page state involves storing small bits of page information in hidden text fields or in the `ViewState`. For instance, Recipe 7.3 shows how a page with multiple states can remember the current state value between `postbacks`. In this instance, information is stored in a hidden field each time a page is submitted to the server so the state can be restored on return to the client

In the chapter's last example, page state is used to store a complex object in the`ViewState` for tracking state information between page submittals. The example demonstrates how you can emulate two-way data binding with the `DataGrid`, a capability that is native to the `GridView` control (see Recipe 2.16) but not to the`DataGrid`. With two-way data binding, any changes made to the data in the bound controls are automatically updated in the underlying data container, making updates to the original data source simple. In web forms, because the connection to the underlying data container is broken when the page is rendered, a bit of additional work is required to update the original data source, which the recipe shows you how to do.

PREV

PREV

Recipe 7.2. Maintaining Information Needed by All Users of an Application

Problem

You want to make certain data available to all users of an application.

Solution

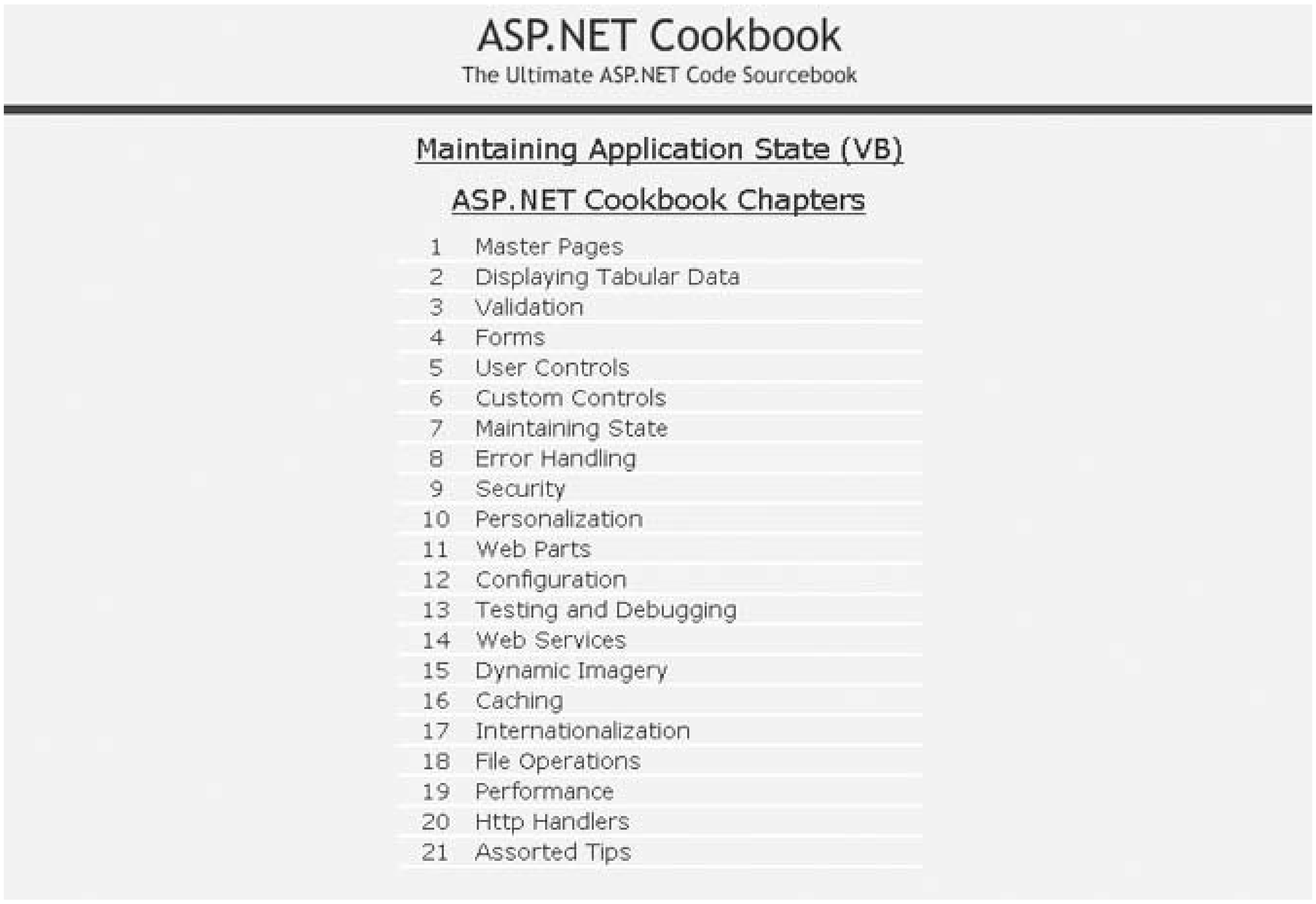
Place the code needed to find and load the data in the `Application_Start` method of *global.asax* and store it in the `Application` object:

In the *global.asax* file, use the .NET language of your choice to:

1. Create an event handler for the `Application_Start` event.
2. Load the application data and store it in the `Application` object.

The code we've written to demonstrate this solution is shown in Examples 7-1, 7-2 , 7-3 , 7-4 , 7-5 , 7-6 through 7-7 . Examples 7-1 and 7-2 show the VB and C# *global.asax* files; this code reads data from a database and places it in the `Application` object. Examples 7-3 and 7-4 show the VB and C# classes for constants used throughout the application to access the data in the `Application` object. Figure 7-1 shows a simple form we've created to view the application state data. Example 7-5shows the *.aspx* file that produces the form. Examples 7-6and 7-7 show the companion VB and C# code-behind files that demonstrate how to access the application state data.

Figure 7-1. A view of some sample application state data



ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Maintaining Application State (VB)

ASP.NET Cookbook Chapters

1	Master Pages
2	Displaying Tabular Data
3	Validation
4	Forms
5	User Controls
6	Custom Controls
7	Maintaining State
8	Error Handling
9	Security
10	Personalization
11	Web Parts
12	Configuration
13	Testing and Debugging
14	Web Services
15	Dynamic Imagery
16	Caching
17	Internationalization
18	File Operations
19	Performance
20	Http Handlers
21	Assorted Tips

Discussion

The purpose of the `Application` object is to store information once that can be simultaneously shared with all users of the application without having to access it repeatedly from a database or some other data store. One example is when you want to store and share the number of times an application has been hit by all the users of the application. Another example is when you want to store and share some common reference information, as illustrated in Figure 7-1. Using the `Application` object provides the ideal means to accomplish these tasks. The `Application` object is similar to the `Session` object (discussed in the next recipe), except that it stores global information as opposed to information about an individual session. The `Application` object is a property of the `Page` object that provides the ability to store almost any data and offer access to it throughout the application by all users. Storing commonly used global data in memory can improve the application's performance.

The `Cache` object can also be used to store data used throughout an application. See Chapter 16 for examples of using the `Cache` object.

The code we've written to illustrate this recipe provides an example of loading information from a database and storing it in the `Application` object so it can be accessed by any page of an application without having to retrieve the data from the database each time the data is needed.

Data that you want to make available throughout the application should be initialized when the application starts. The `Application_Start` event handler in the *global.asax* file class is the best place to do this because the `Application_Start` event is raised the first time the application is accessed.

In our example, the `Application_Start` event handler reads the titles of the chapters of this book from a database into a `DataTable` and stores the table of chapter data in the `Application` object.

Data stored in the `Application` object can be accessed from any page of an application. When you access the data, you must cast it to the correct type because all data is stored in the `Application` object as `Objects` ; without properly casting the data, your code will not compile. Our example retrieves the data and binds it to a `Repeater` control in the `Page_Load` method in Examples 7-6 (VB) and 7-7 (C#). (Data binding is described in Chapter 2.)

> To avoid "hardcoding" names of variables placed in the `Application` object, we recommend you define public constants in a class that is accessible throughout your application, such as `appconstants` in Examples 7-3 (VB) and 7-4 (C#). In ASP.NET 1.x, these constants could be placed in the `Global.asax.vb` (or `Global.asax.cs`) class. With the changes in ASP.NET 2.0 to use partial classes, these constants must be placed in another class to provide application-wide access.
>
> On another note, any object placed in the `Application` object must be free-threaded; otherwise, deadlocks, race conditions, and access violations can occur. Any object created from the classes in the Common Language Runtime is free-threaded and can be safely stored in the `Application` object. VB6 objects are apartment-threaded and should not be placed in the `Application` object.

You can update data stored in the `Application` object whenever your application requires it. ASP.NET is multithreaded, so many threads can access the variables within the scope of an application at the same time. Changing the variables' values in an uncontrolled fashion can cause data concurrency issues. To prevent contention between threads, the `Application` object must be able to block access to itself while changes are being made. You can do this by calling the Lock method of the `Application` object, making your changes, and calling the `Unlock` method, as shown here:

```
Application.Lock()
Application.Add(APP_CHAPTER_DATA, _
   ds.Tables(CHAPTER_TABLE))
Application.UnLock()

Application.Lock();
Application.Add(APP_CHAPTER_DATA,
   ds.Tables[CHAPTER_TABLE]);
Application.UnLock();
```

Your application should always minimize the time the`Application` object is locked because all other threads are held off until the lock is released. To avoid permanent locks, ASP.NET automatically performs the unlock operation when a request is completed, a request times out, or an unhandled exception occurs and causes the request to fail.

Whenever you use the `Application` object in an application, consider the following points:

- The `Application` object is not shared across servers in a web farm. Each server maintains its own copy of the `Application` object. If the values stored in the application are different on each of the servers in the web farm, your application may operate differently depending on which server was accessed.
- Any data stored in the `Application` object is cleared when the application restarts. If any of that data needs to be persisted, you can place code in the`Application_End` event handler of *global.asax* to store the data in a database or the like. Be careful in relying on the `Application_End` event handler to persist any important data. It is not called if the application stops abnormally, such as in a power failure or an application crash.

See Also

Chapter 16 for information on using the Cache object

Example 7-1. Maintaining application state (global.asaxVB)

```
<%@ Application Language="VB" %>
<%@ Import Namespace="System.Configuration.ConfigurationManager" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>
<%@ Import Namespace="ASPNetCookbook.VBExamples" %>
<script RunAt="server">

 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the application start
 ''' event. It is responsible for initializing application variables.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
  Dim dbConn As OleDbConnection = Nothing
  Dim da As OleDbDataAdapter = Nothing
  Dim dTable As DataTable = Nothing
  Dim strConnection As String
  Dim cmdText As String
```

```
   Try
    'get the connection string from web.config and open a connection
    'to the database
    strConnection = _
     ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDbConnection(strConnection)
    dbConn.Open()

    'build the query string and get the data from the database
    cmdText = "SELECT ChapterNumber, Title, Description " & _
       "FROM Chapter " & _
       "ORDER BY ChapterNumber"

    'fill the dataset with the chapter data
    da = New OleDbDataAdapter(cmdText, dbConn)
    dTable = New datatable
    da.Fill(dtable)

    'store the table containing the chapter data in the Application object
    Application.Add(appconstants.APP_CHAPTER_DATA, _
     dtable)

   Finally
    If (Not IsNothing(dbConn)) Then
     dbConn.Close()
    End If
   End Try
  End Sub
</script>
```

Example 7-2. Maintaining application state (global.asaxC#)

```
<%@ Application Language="C#" %>
<%@ Import Namespace="System.Configuration" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>
<%@ Import Namespace="ASPNetCookbook.CSExamples" %>

<script RunAt="server">

 ///***********************************************************************
 /// <summary>
 /// This routine provides the event handler for the application start
 /// event. It is responsible for initializing application variables.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 void Application_Start(Object sender, EventArgs e)
 {
  OleDbConnection dbConn = null;
  OleDbDataAdapter da = null;
  DataTable dTable = null;
  String strConnection = null;
  String cmdText = null;

  try
  {
   // get the connection string from web.config and open a connection
   // to the database
   strConnection = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
   dbConn = new OleDbConnection(strConnection);
   dbConn.Open();

   // build the query string and get the data from the database
   cmdText = "SELECT ChapterNumber, Title, Description " +
      "FROM Chapter " +
      "ORDER BY ChapterNumber";

   // fill the dataset with the chapter data
   da = new OleDbDataAdapter(cmdText, dbConn);
   dTable = new DataTable();
   da.Fill(dTable);

   // store the table containing the chapter data in the Application object
   Application.Add(appconstants.APP_CHAPTER_DATA,
    dTable);
   } // try

  finally
  {
   // cleanup
   if (dbConn != null)
   {
    dbConn.Close();
   }
  } // finally
 } // Application_Start
</script>
```

Example 7-3. Application constants (.vb)

```
Option Explicit On
Option Strict On

Imports System.Drawing

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class constants that are used throughout the application
 ''' </summary>
 Public Class appconstants
  'the following constant used to define the name of the variable used to
  'store the chapter data in the application object
  Public Const APP_CHAPTER_DATA As String = "ChapterData"

 End Class 'appconstants
End Namespace
```

Example 7-4. Application constants (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class constants that are used throughout the application
 /// </summary>
 public class appconstants
 {
  // the following constant used to define the name of the variable used to
  // store the chapter data in the application object
  public const String APP_CHAPTER_DATA = "ChapterData";

 } // appconstants
}
```

Example 7-5. Using application state data (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH07ApplicationStateVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH07ApplicationStateVB"
 Title="Application State" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Maintaining Application State (VB)
 </div>
 <table width="40%" border="0" align="center"
     cellpadding="0" cellspacing="0">
  <thead>
   <tr>
    <th colspan="2"
       class="pageHeading">ASP.NET Cookbook Chapters</th>
   </tr>
  </thead>
  <tr>
    <td><img src="images/spacer.gif" height="10" border="0" alt=""/></td>
  </tr>
  <asp:Repeater id="repMenuItems" runat="server">
   <ItemTemplate>
    <tr class="labelText">
     <td align="center" width="15%">
    <%#Eval("ChapterNumber")%></td>
     <td width="85%">
    <%#Eval("Title")%></td>
    </tr>
    <tr>
     <td bgcolor="#FFFFFF" colspan="2">
     <img src="images/spacer.gif" border="0" height="2" alt=""/></td>
    </tr>
   </ItemTemplate>
  </asp:Repeater>
 </table>
</asp:Content>
```

Example 7-6. Using application state data code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System.Data

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH07ApplicationStateVB.aspx
 ''' </summary>
 Partial  Class  CH07ApplicationStateVB
  Inherits System.Web.UI.Page
  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim chapterData As DataTable
   If (Not Page.IsPostBack()) Then
    'get the chapter data stored in the application object
    chapterData = CType(Application.Item(appconstants.APP_CHAPTER_DATA), _
      DataTable)

    'bind it to the repeater to display the chapter data
    repMenuItems.DataSource = chapterData
    repMenuItems.DataBind()
   End If
  End Sub 'Page_Load
 End  Class  'CH07ApplicationStateVB
End  Namespace
```

Example 7-7. Using application state data code-behind (.cs)

```
using System;
using System.Data;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH07ApplicationStateCS.aspx
 /// </summary>
 public partial class CH07ApplicationStateCS : System.Web.UI.Page
 {
   ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   DataTable chapterData = null;

   if (!Page.IsPostBack)
   {
    // get the chapter data stored in the application object
    chapterData = (DataTable)(Application[appconstants.APP_CHAPTER_DATA]);

    // bind it to the repeater to display the chapter data
    repMenuItems.DataSource = chapterData;
    repMenuItems.DataBind();
   }
  } // Page_Load
 } // CH07ApplicationStateCS
}
```

Recipe 7.3. Maintaining Information About a User Throughout a Session

Problem

You want to make personalized information available to the users of your application for as long as each remains active without accessing a database each time the information is needed and regardless of the number of pages traversed.

Solution

Create a class in which to store the personalized data, instantiate the class and load the data, store the data object in the `Session` object, and access the data from the `Session` object as required.

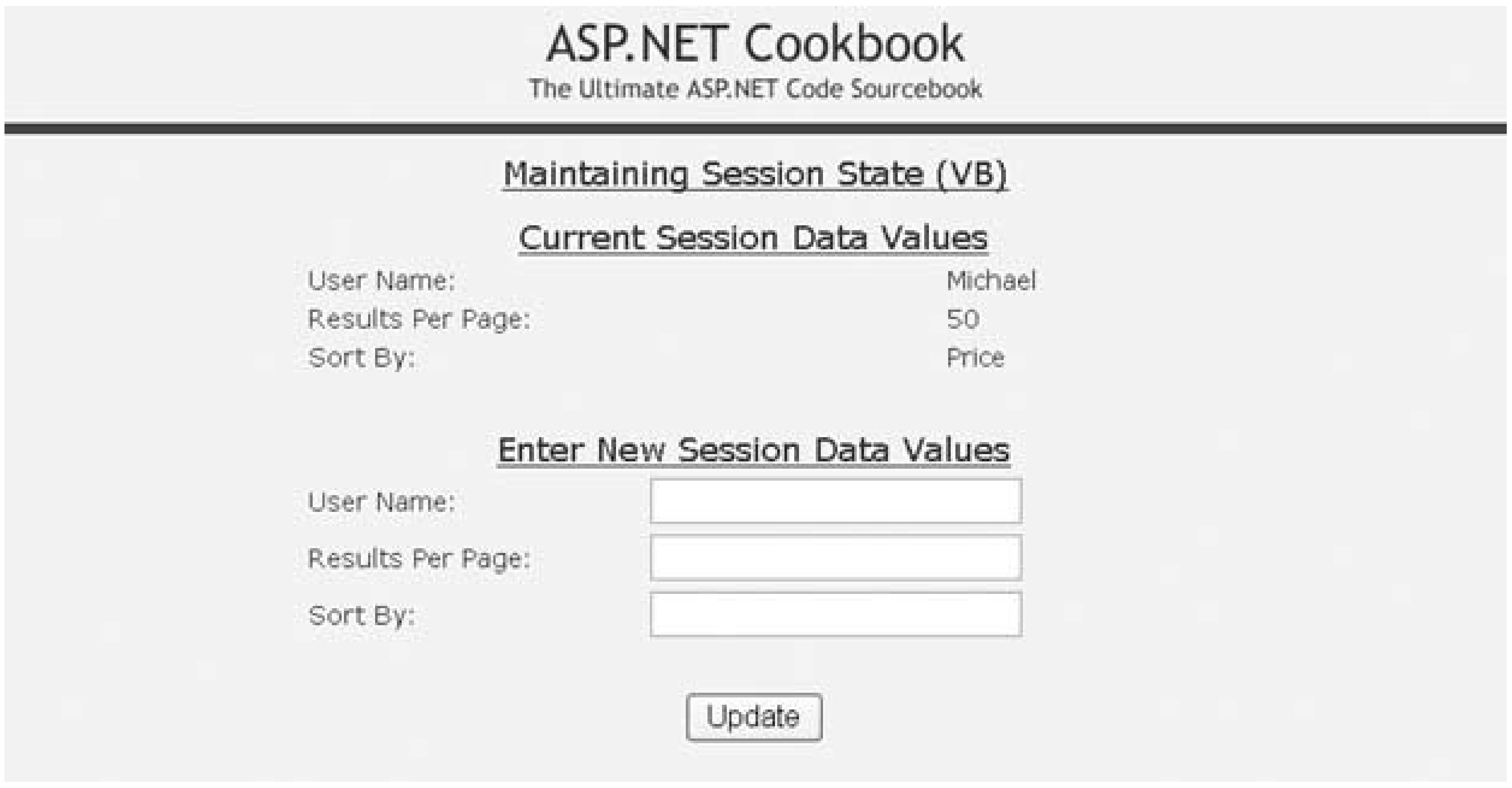
In the code-behind class for your ASP.NET pages that need access to the data, use the .NET language of your choice to:

1. Check to see if the object used to store the personalized data exists in the `Session` object.
2. If the object exists, retrieve the object from `Session`. If the object does not exist, instantiate the class used for the personalized data and store it in the `Session` object.
3. Use the data as required in your application.

A simple example that illustrates this solution is shown in Examples 7-8, 7-9, 7-10, 7-11 through 7-12. The example uses the class shown in Examples 7-8 (`CH07PersonalDataVB` for VB) and 7-9 (`CH07PersonalDataCS` for C#) to provide a container for some simple personalized data. This class contains properties for each of the data items and a default constructor.

Figure 7-2 shows a form that we've created for viewing the current contents of the personalized data stored in the `Session` object and for entering new session state data values. Example 7-10 shows the *.aspx* file that produces the form. Examples 7-11 and 7-12 show the companion VB and C# code-behind files.

Figure 7-2. Form for viewing and entering session state data values


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Maintaining Session State (VB)

Current Session Data Values

User Name:	Michael
Results Per Page:	50
Sort By:	Price

Enter New Session Data Values

User Name:	
Results Per Page:	
Sort By:	

Discussion

An approach we like for maintaining personalized data about auser for the duration of a sessionan approach often referred to as *personalization*is to create a class to hold the data, instantiate and populate the object, store the object in the `Session` object, and access the data from the `Session` object.

> This recipe takes a slightly different approach topersonalization from that found in the recipes in Chapter 10, which leverage ASP.NET 2.0's built-in profile features. Here the emphasis is on relatively short-lived datathat is, data that must remain available while the user is active. What's more, your code must undertake all the necessary steps to instantiate and populate an object to hold personalized settings, store it in session state, and access it from session state when needed. By contrast, when using ASP.NET 2.0's built-in profile features, much of the mechanics of persisting information is handled for you automatically and transparently using the profile framework. More important, ASP.NET 2.0's profile features are geared more toward long-lived data, such as a user's name, address, and other user-specific information. These customizations can be persisted and made available to the user on subsequent visits to the site. This forms the basis of ASP.NET 2.0's new profile features and is the focus of the recipes in Chapter 10.

As its name implies, you can use the `Session` object in ASP.NET to store information needed for a particular user session. Variables stored in the `Session` object are not discarded when the user navigates between the pages of an application. Rather, they are persisted for the entire session.

To illustrate this approach, the code-behind in our example contains the logic needed to access the data in the Session object. We make use of the Page_Load method in Examples 7-11 (VB) and 7-12 (C#) to check if the Session object contains the personalization data. If not, we create a new CH07PersonalDataVB (VB) or CH07PersonalDataCS (C#) object using default values, and store this new data object in the Session object associated with the current session. Otherwise, we retrieve a reference to the object containing the personalized data. Finally, we update the contents of the form by passing the personalization object to a method that uses the data to update the contents of the form.

In Examples 7-11 (VB) and 7-12 (C#), a constant, SES_PERSONALIZATION_DATA, is used to define the name of the variable placed in the Session object. This is done to avoid having to hardcode the name of the variable in multiple locations in the code. In an application where the data is accessed in multiple pages, the constant should be stored in another class containing global constants as described in Recipe 7.1.

For this example, the personalized data is updated when the user enters new personalization data values and clicks the Update button. In the button click event handler, we check whether the data has been stored in the Session object, using the same code we used in the Page_Load method. You should always check this condition to avoid the error that will be thrown if the data is no longer in the Session object. Loss of data can occur if the session times out and ASP.NET deletes all variables for the user session.

Next, we update the contents of the personalization object with data from the form. In a production setting, your code will need to perform validation on the data to ensure that it is of the correct type, in the correct form, within the correct range, and so on. Finally, we store the personalization data in the Session object and update the form contents.

This example is simple, but it shows the mechanics associated with storing and retrieving data in the Session object. In a full application, the personalization data could be read from a database when the user logs in. If your application uses cookies to identify a user, the Session_Start event handler of *global.asax* provides an ideal place to retrieve the cookie that identifies the user, get the user's personalization data from a database, and place the data in the Session object.

You can place any object in the Session object, but take care not to overuse the Session object. Large objects can impact the application's performance by tying up system resources.

To provide the ability to associate session data with a specific user, ASP.NET assigns a session ID to each user session. The session ID is then used by ASP.NET to retrieve the Session object associated with the user requesting a page.

The HTTP protocol is stateless, so some method of associating the incoming requests with a specific user is required. By default, ASP.NET sends a cookie containing the session ID to the client browser as part of its response to the first page request by a user. Subsequent page requests return the cookie data. ASP.NET retrieves the cookie data from the request, extracts the session ID, retrieves the Session object for the specific user, and processes the requested page.

The cookie sent to the client browser is an in-memory cookie and is not persisted on the client machine. If the user closes the browser, the cookie containing the session ID will be destroyed.

To support applications that do not use cookies, ASP.NET provides the ability to modify the URL automatically to contain the session ID (called *URL munging*). This method of tracking the session ID is configured in the *web.config* file and is discussed in Recipe 12.4.

ASP.NET supports three methods of storing the session information. By default, the session data is stored in memory within the ASP.NET process. The session data can also be stored in memory in an out-of-process state server or SQL Server. The storage methods are configured in the *web.config* file and are discussed in Recipe 12.4.

When using the `Session` object in your application, consider the following points:

- By default, a session times out after 20 minutes of inactivity. When a session times out, the ASP.NET process destroys all session data, and the resources used by the session variables are recovered. The session timeout is configured in the *web.config* file and is discussed in Recipe 12.4.
- If any special operations are required when a user session ends, they can be performed in the `Session_End` event handler of *global.asax*. This event is raised whenever a session ends, whether it is done programmatically or because the session times out. However, the `Session_End` event may not be raised if ASP.NET is terminated abruptly.

See Also

Recipes 7.1, 12.4, and Chapter 10 for ASP.NET 2.0's built-in personalization features

Example 7-8. Class used to store data in session object (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the container to store personalization data
 ''' for a user
 ''' </summary>
 Public Class CH07PersonalDataVB
  'private attributes with default values
  Private mUsername As String = ""
  Private mResultsPerPage As Integer = 25
  Private mSortBy As String = "Title"

  '''*********************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the username data
  ''' </summary>
  Public Property username() As String
   Get
    Return (mUsername)
   End Get
```

```
    Set(ByVal Value As String)
     mUsername = Value
    End Set
   End Property 'username

   '''**********************************************************************
   ''' <summary>
   ''' This property provides the ability to get/set the resultsPerPage data
   ''' </summary>
   Public Property resultsPerPage() As Integer
    Get
     Return (mResultsPerPage)
    End Get
    Set(ByVal Value As Integer)
     mResultsPerPage = Value
    End Set
   End Property 'resultsPerPage

   '''**********************************************************************
   ''' <summary>
   ''' This property provides the ability to get/set the sortBy data
   ''' </summary>
   Public Property sortBy() As String
    Get
     Return (mSortBy)
    End Get
    Set(ByVal Value As String)
     mSortBy = Value
    End Set
   End Property 'sortBy
  End Class 'CH07PersonalDataVB
End Namespace
```

Example 7-9. Class used to store data in session object (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the container to store personalization data
  /// for a user
  /// </summary>
  public class CH07PersonalDataCS
  {
   // private attributes with default values
   private String mUsername = "";
   private int mResultsPerPage = 25;
   private String mSortBy = "Title";

   ///***********************************************************************
   /// <summary>
   /// This property provides the ability to get/set the username data
   /// </summary>
   public String username
   {
    get
    {
     return (mUsername);
    }
    set
    {
     mUsername = value;
    }
   } // username

   ///***********************************************************************
   /// <summary>
   /// This property provides the ability to get/set the resultsPerPage data
   /// </summary>
   public int resultsPerPage
   {
    get
    {
     return (mResultsPerPage);
    }
    set
    {
     mResultsPerPage = value;
    }
   } // resultsPerPage

   ///***********************************************************************
   /// <summary>
   /// This property provides the ability to get/set the sortBy data
   /// </summary>
   public String sortBy
   {
    get
    {
     return (mSortBy);
    }
    set
    {
     mSortBy = value;
    }
   } // mSortBy
  } // CH07PersonalDataCS
 }
```

Example 7-10. Maintaining user state (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH07SessionStateVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH07SessionStateVB"
 Title="State Session" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Maintaining Session State (VB)
 </div>
 <table width="60%" align="center">
  <tr>
    <td colspan="2" align="center" class="pageHeading">
    Current Session Data Values</td>
  </tr>
  <tr class="labelText">
   <td>User Name: </td>
   <td><asp:Label ID="labUserName" Runat="server" /></td>
  </tr>
  <tr class="labelText">
   <td>Results Per Page: </td>
   <td><asp:Label ID="labResultsPerPage" Runat="server" /></td>
  </tr>
  <tr class="labelText">
   <td>Sort By: </td>
   <td><asp:Label ID="labSortBy" Runat="server" /></td>
  </tr>
 </table>
 <br />
 <table width="60%" align="center">
  <tr>
    <td colspan="2" align="center" class="pageHeading">
    Enter New Session Data Values</td>
  </tr>
  <tr class="labelText">
   <td>User Name: </td>
   <td><asp:TextBox ID="txtUserName" Runat="server" /></td>
  </tr>
  <tr class="labelText">
   <td>Results Per Page: </td>
   <td><asp:TextBox ID="txtResultsPerPage" Runat="server" /></td>
  </tr>
  <tr class="labelText">
   <td>Sort By: </td>
   <td><asp:TextBox ID="txtSortBy" Runat="server" /></td>
  </tr>
  <tr>
   <td colspan="2" align="center">
    <br />
    <asp:Button ID="btnUpdate" runat="server"
       Text="Update"
       OnClick="btnUpdate_Click" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 7-11. Maintaining user state code-behind (.vb)

```

Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' CH07SessionStateVB.aspx
 ''' </summary>
 Partial Class CH07SessionStateVB
  Inherits System.Web.UI.Page

  'The following constant defines the name of the session variable used
  'to store the user personalization data
  Public Const SES_PERSONALIZATION_DATA As String = "PersonalizationData"

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim personalData As CH07PersonalDataVB

   If (Not Page.IsPostBack()) Then
    'check to see if the session data exists
    If (IsNothing(Session(SES_PERSONALIZATION_DATA))) Then
    'data does not exist in session so create a new personalization
    'object and place it in session scope
    personalData = New CH07PersonalDataVB
    Session.Add(SES_PERSONALIZATION_DATA, _
     personalData)
    Else
    'data exists in session so get a reference to the data
    personalData = CType(Session(SES_PERSONALIZATION_DATA), _
        CH07PersonalDataVB)
    End If

    'update contents on the form
    updateFormData(personalData)
   End If
  End Sub 'Page_Load

   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the update button click
  ''' event. It is responsible for updating the contents of the session
  ''' variable used to store the personalization data and updating the form.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnUpdate_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
   Dim personalData As CH07PersonalDataVB
   'check to see if the session data exists
   If (IsNothing(Session(SES_PERSONALIZATION_DATA))) Then
    'data does not exist in session so create a new personalization object
    personalData = New CH07PersonalDataVB
   Else
    'data exists in session so get a reference to the data
    personalData = CType(Session(SES_PERSONALIZATION_DATA), _
       CH07PersonalDataVB)
   End If

   'update contents of session object from form
   personalData.username = txtUserName.Text
   personalData.resultsPerPage = CInt(txtResultsPerPage.Text)
   personalData.sortBy = txtSortBy.Text

   'update contents of session object
   Session(SES_PERSONALIZATION_DATA) = personalData

   'update contents on the form
   updateFormData(personalData)
  End Sub 'btnUpdate_Click

   '''*************************************************************************
  ''' <summary>
  ''' This routine updates the contents of the form from the passed data.
  ''' </summary>
  '''
  ''' <param name="personalData">Set the the personalization data used
```

```
  ''' to update the data on the form
  ''' </param>
  Private Sub updateFormData(ByVal personalData As CH07PersonalDataVB)
   labUserName.Text = personalData.username
   labResultsPerPage.Text = personalData.resultsPerPage.ToString()
   labSortBy.Text = personalData.sortBy
  End Sub 'update
   End Class 'CH07 SessionStateVB
End Namespace
```

Example 7-12. Maintaining user state code-behind (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH07SessionStateCS.aspx
 /// </summary>
 public partial class CH07SessionStateCS : System.Web.UI.Page
 {
  // The following constant defines the name of the session variable used
  // to store the user personalization data
  public const String SES_PERSONALIZATION_DATA = "PersonalizationData";
  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   CH07PersonalDataCS personalData = null;

   if (!Page.IsPostBack)
   {
    // check to see if the session data exists
    if (Session[SES_PERSONALIZATION_DATA] == null)
    {
    // data does not exist in session so create a new personalization
    // object and place it in session scope
    personalData = new CH07PersonalDataCS();
    Session.Add(SES_PERSONALIZATION_DATA,
     personalData);
    }
    else
    {
    // data exists in session so get a reference to the data
    personalData = (CH07PersonalDataCS)
        (Session[SES_PERSONALIZATION_DATA]);
    }

    // update contents on the form
    updateFormData(personalData);
   }
  } // Page_Load

  ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the update button click
  /// event. It is responsible for updating the contents of the session
  /// variable used to store the personalization data and updating the form.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnUpdate_Click(object sender, EventArgs e)
  {
   CH07PersonalDataCS personalData = null;

   //check to see if the session data exists
   if (Session[SES_PERSONALIZATION_DATA] == null)
   {
    // data does not exist in session so create a new
    // personalization object
    personalData = new CH07PersonalDataCS();
   }
   else
   {
    //data exists in session so get a reference to the data
    personalData = (CH07PersonalDataCS)(Session[SES_PERSONALIZATION_DATA]);
   }

   // update contents of session object from form
   personalData.username = txtUserName.Text;
   personalData.resultsPerPage = Convert.ToInt32(txtResultsPerPage.Text);
   personalData.sortBy = txtSortBy.Text;

   //update contents of session object
   Session[SES_PERSONALIZATION_DATA] = personalData;

   // update contents on the form
   updateFormData(personalData);
  } // btnUpdate_Click

  ///*************************************************************************
  /// <summary>
  /// This routine updates the contents of the form from the passed data.
  /// </summary>
   private void updateFormData(CH07PersonalDataCS personalData)
  {
   labUserName.Text = personalData.username;
   labResultsPerPage.Text = personalData.resultsPerPage.ToString();
   labSortBy.Text = personalData.sortBy;
  } // updateFormData
 } // CH07SessionStateCS
}
```

PREV

Recipe 7.4. Preserving Information Between Postbacks

Problem

You have a page that needs to remember the state's value between `postbacks` to determine how to display the page.

Solution

Use the `RegisterHiddenField` method of the `ClientScript` object to create a hidden text field in the rendered page.

Nothing special is required in the *.aspx* file. Instead, in the code-behind class for the page, use the .NET language of your choice to:

1. Programmatically insert a hidden text field into the form using the`RegisterHiddenField` method of the `ClientScript` object.
2. Use this field to store the value of the state you wish to preserve between `postbacks`.
3. Access the hidden text field on subsequent submissions of the page to the server.

Figure 7-3 shows the output of a form that preserves the page state using a hidden field. Clicking the Prev/Next buttons decrements/increments the value in the hidden field by one.Example 7-13 shows the *.aspx* file that produces the form. Examples 7-14 and 7-15 show the companion VB and C# code-behind files that demonstrate how to access the application state data.

Figure 7-3. Maintaining page state with a hidden field

Discussion

An approach we favor for remembering the current state of a value between postbacks to the server involves programmatically inserting one or more hidden text fields into a form.

A similar technique is commonly used by classic ASP developers, who explicitly place a hidden text field in the form and set its value in the page code. You can use the same technique in ASP.NET pages. However, ASP.NET, unlike classic ASP, lets you programmatically insert hidden text fields into a form at runtime. The significant advantage here is that all code you write is kept in the code-behind, allowing the *.aspx* file to contain only the presentation aspects (hidden fields contain no user interface).

One negative aspect of using hidden fields for storing state data is that the data is stored in plain text. If you do not want the data to be visible in the rendered HTML, you should store the data in the ViewState instead, as described in Recipe 7.4.

As an example of when you might use hidden fields, suppose your application supports complex sorting within a DataGrid, such as a two-way sort that involves ascending and descending columns and might even be supported by your own sorting expressions. You can use hidden fields to maintain information about how a user has performed a specific sort so you can preserve the user's sorting order and choice of sorted columns.

Our example that illustrates the solution is simple to focus on the concept of storing information in hidden fields each time a page is submitted to the server. The example programmatically inserts a single hidden field into a form and then increments or decrements itsvalue based on the button clicked by the user.

Because the hidden field is accessed from many points in the code-behind, a constant at the class level, PAGE_STATE, defines the name of the hidden field that is programmatically inserted into the rendered page. This avoids hardcoding the hidden field name throughout the code and the associatec maintenance issues.

In the Page_Load method, updatePage, which is the method used to update the page based on the page state, is called passing a value of 0 as the initial page state value.

In the updatePage method, a label is updated to indicate the current state value. After updating the page to reflect the current page state, the RegisterHiddenField method of the Page object is used to save the page state value in the rendered page.

The rendered page contains the following hidden form variable that can be retrieved when the page i submitted to the server. The name is set to the value of the PAGE_STATE constant, and the value is set to the current page state:

```
<input type="hidden" name="PageState" id="PageState" value="0" />
```

Hidden fields always store the value as a string. This requires any data saved using the technique to be converted to a string when the `RegisterHiddenField` method is called.

In this example, two buttons are provided to increment and decrement the page state, respectively. An event handler is added in the code-behind for each of the button click events. The event handler for the increment button is called `btnNextState_Click`; the event handler for the decrement button is `btnPrevState_Click`. In the event handlers, the current page state is retrieved from the hidden field, adjusted as required, and then the `updatePage` method is called to update the page and save the new page state value in the form when the page is rendered.

The example demonstrates how easy ASP.NET has made it to persist information needed for each round trip between the server and the client. For an example of persisting more complex data between page submittals, refer to Recipe 7.4.

See Also

Recipe 7.4

Example 7-13. Maintaining page state with hidden values (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH07HiddenValuesVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH07HiddenValuesVB"
 Title="Hidden Values" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Maintaining Page State With Hidden Fields (VB)
 </div>
 <table width="30%" align="center" id="tabState" runat="server">
  <tr class="labelText">
   <td>Current Page State: </td>
   <td><asp:Label ID="labPageState" Runat="server" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
     <br />
     <asp:Button ID="btnPrevState" runat="server"
        Text="Prev State"
        OnClick="btnPrevState_Click" />
     <asp:Button ID="btnNextState" runat="server"
        Text="Next State"
        OnClick="btnNextState_Click" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 7-14. Maintaining page state with hidden values code-behind (.vb)

```

Option Explicit
On Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH07HiddenValuesVB.aspx
 ''' </summary>
 Partial Class CH07HiddenValuesVB
  Inherits System.Web.UI.Page

  'The following variable defines the name of the hidden field in the
  'form used to track the page state
  Private PAGE_STATE As String = "PageState"

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   If (Not Page.IsPostBack()) Then
    'not a postback so initialize the page state to 0
    updatePage(0)
   End If
  End Sub 'Page_Load

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the next state button
  ''' click event. It is responsible for setting the page state ahead
  ''' one state.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
```

```
   Protected Sub btnNextState_Click(ByVal sender As Object, _
         ByVal e As System.EventArgs)
    Dim pageState As Integer

    pageState = CInt(Request.Form(PAGE_STATE))
    pageState += 1
    updatePage(pageState)
   End Sub 'btnNextState_Click

   '''*************************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the previous state button
   ''' click event. It is responsible for setting the page state back one
   ''' state.
   ''' </summary>
   '''
   ''' <param name="sender">Set to the sender of the event</param>
   ''' <param name="e">Set to the event arguments</param>
   Protected Sub btnPrevState_Click(ByVal sender As Object, _
         ByVal e As System.EventArgs)
    Dim pageState As Integer

    pageState = CInt(Request.Form(PAGE_STATE))
    pageState -= 1
    updatePage(pageState)
   End Sub 'btnPrevState_Click

   '''*************************************************************************
   ''' <summary>
   ''' This routine updates the page for the passed page state.
   ''' </summary>
   '''
   ''' <param name="pageState">Set to the current page state</param>
   Private Sub updatePage(ByVal pageState As Integer)

    'update the current page state display
    labPageState.Text = pageState.ToString()

    'register the hidden field used to persist the current page state
    ClientScript.RegisterHiddenField(PAGE_STATE, _
          pageState.ToString())
   End Sub 'updatePage
  End Class 'CH07HiddenValuesVB
 End Namespace
```

Example 7-15. Maintaining page state with hidden values code-behind (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH07HiddenValuesCS.aspx
 /// </summary>
 public partial class CH07HiddenValuesCS : System.Web.UI.Page
 {
  // The following variable defines the name of the hidden field in the
  // form used to track the page state
  private String PAGE_STATE = "PageState";

   ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   if (!Page.IsPostBack)
   {
    // not a postback so initialize the page state to 0
    updatePage(0);
   }
  } // Page_Load

   ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the next state button
  /// click event. It is responsible for setting the page state ahead
  /// one state.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnNextState_Click(object sender, EventArgs e)
  {
   int pageState;

   pageState = Convert.ToInt32(Request.Form[PAGE_STATE]);
   pageState += 1;
   updatePage(pageState);
  } // btnNextState_Click

   ///***********************************************************************
  /// <summary>
```

```
   /// This routine provides the event handler for the previous state button
   /// click event. It is responsible for setting the page state back one
   /// state.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void btnPrevState_Click(object sender, EventArgs e)
   {
    int pageState;

    pageState = Convert.ToInt32(Request.Form[PAGE_STATE]);
    pageState -= 1;
    updatePage(pageState);
   } // btnPrevState_Click

   ///*************************************************************************
   /// <summary>
   /// This routine updates the page for the passed page state.
   /// </summary>
   ///
   /// <param name="pageState">Set to the current page state</param>
   private void updatePage(int pageState)
   {
    // update the current page state display
    labPageState.Text = pageState.ToString();
    // register the hidden field used to persist the current page state
    ClientScript.RegisterHiddenField(PAGE_STATE,
          pageState.ToString());
   } // updatePage
  } // CH07HiddenValuesCS
 }
```

Recipe 7.5. Preserving Information Across Multiple Requests for a Page

Problem

You have a page that contains complex object information you need to preservebetween requests for the page. The data contains information you want to be unreadable in the rendered HTML, and you do not want to use a database to preserve the information.

Solution

Use the `ViewState` property of the `Page` object to store the data. You can access the data when the page is submitted back to the server.

In the code-behind class for the page, use the .NET language of your choice to add all the code necessary to handle the storage and recovery of the object data to and from the`ViewState`.

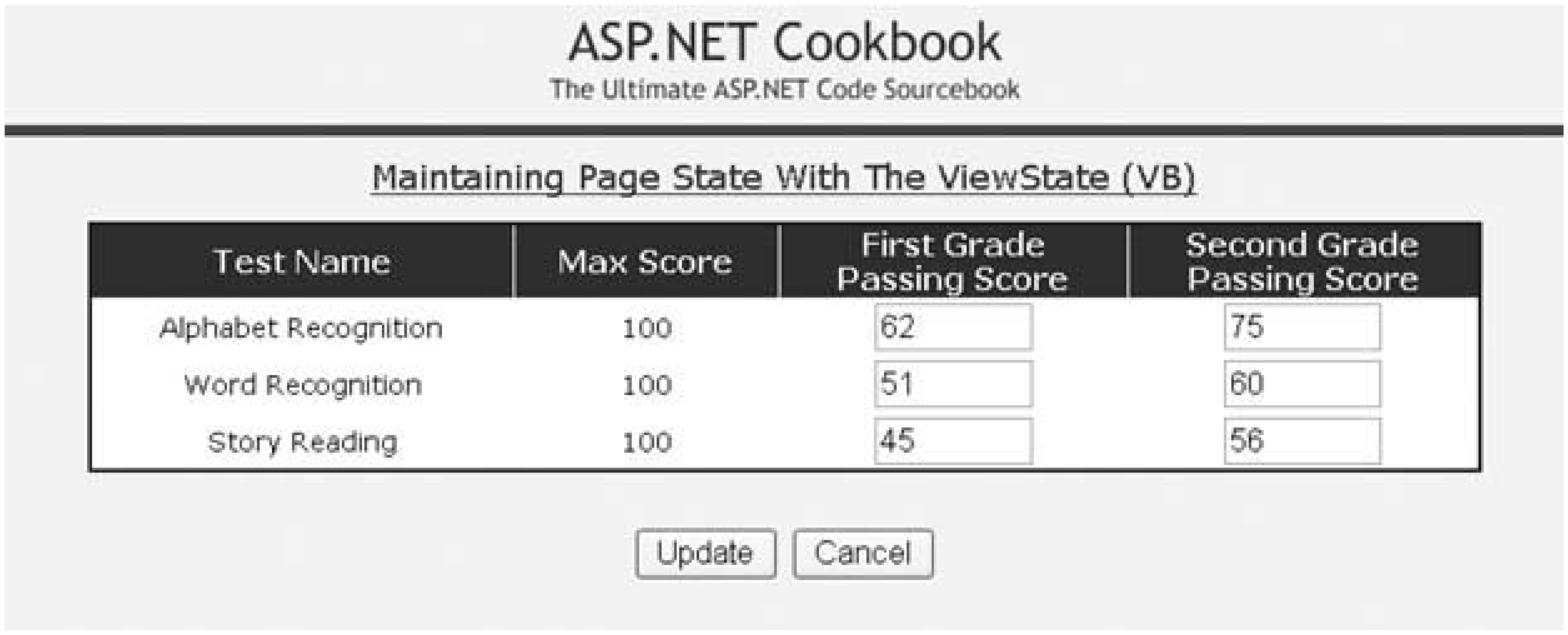
In a separate class file, use the .NET language of your choice to define the container in which you wil store the data in the `ViewState`.

The application that illustrates this solution is shown inExamples 7-16, 7-17, 7-18, 7-19 through 7-20. Example 7-16 shows the *.aspx* file. Examples 7-17 and 7-18 show the VB and C# code-behind files. Examples 7-19 and 7-20 show the VB and C# data service class. Figure 7-4 shows the form produced by the application.

Discussion

The `ViewState` is an object similar to the `Application` and `Session` objects discussed in the previous recipes; however, its method of data storage is different. Unlike the`Application` and `Session` objects, which are stored in server memory, the `ViewState` is stored in a hidden form field within the HTML sent to the browser. This property lets you store page state information directly in a page and then retrieve it when the page is posted back to the server without using server resources. This technique does result, however, in additional data being transmitted to and from the server.

Figure 7-4. Maintaining page state with ViewState


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Maintaining Page State With The ViewState (VB)

Test Name	Max Score	First Grade Passing Score	Second Grade Passing Score
Alphabet Recognition	100	62	75
Word Recognition	100	51	60
Story Reading	100	45	56

ASP.NET uses the ViewState to store state information for the server controls on your form so it can rehydrate (or deserialize the data for) the controls upon submittal of the page to the server. You can use the ViewState for storing page state data in your application as well. What we mean by "page state data" in this context is user-specific state values not stored by a control. Values are tracked in ViewState similarly to how they are tracked in Session (described in Recipe 7.2) and Cache (described in Recipe 16.7).

An example of when you might want to use ViewState in this way is when you want to display a list o items in a DataGrid, and each user wants to sort the DataGrid's columns differently. In this context, the sort order is a small piece of user-specific page state that you want to maintain when the page is submitted back to the server. ViewState is a fine place to store this type of value, and the Visual Studio help files are replete with examples of this sort.

The example we've written to illustrate this solution is more ambitious than the previous example, fairly long, but worth the effort. It demonstrates the ability to store complex objects in the ViewState for tracking state information between page submittals and how to emulate the two-way data bindin available with the GridView control but not available with the DataGrid control. When used together, these two concepts provide the ability to use many of the features of ASP.NET, which can simplify th code required to develop an application.

> If you find yourself using the DataGrid because it's the best choice for your application, this recipe will help you implement two-way data binding with it. The GridView, however, inherently supports two-way data binding.

The *.aspx* file, shown in Example 7-16, is typical of a page containing a data-bound DataGrid (see Chapter 2). The one difference in this example is the use of constants from the data service class to define the fields bound to the columns in the DataGrid. The constants are described later, during the discussion of the data service class.

In Example 7-16, the following change is required to use the *.aspx* file with C#:

- Change the namespace in the imports statement to `ASPNetCookbook.CSExamples`.

You'll find all code that handles the persistence and recovery of the object data you store in the `ViewState` in the code-behind shown in Examples 7-17 (VB) and 7-18 (C#).

The data service class, shown in Examples 7-19 (VB) and 7-20 (C#), provides a container for persisting the data in the `ViewState`. (The data service class is described at the end of this example.)

Because you will want to access the object persisted in the `ViewState` from multiple places in your code, we've defined a constant in the code-behind, `VS_TEST_DATA`, to define the name of the variable used to access the object in the `ViewState`. With this approach, you avoid the problems when you hardcode a variable name throughout your code.

In the `Page_Load` method of the code-behind, an instance of the data service class is created by calling its constructor. The new object is passed to the `bindData` method to bind the data in the object to the `DataGrid` in the *.aspx* file.

Create the new object only when the page is originally rendered. Creating it on subsequent `postbacks` would result in loss of the data entered by the user.

The `bindData` method performs two operations. First, it binds the data in the passed `testData` object to the `DataGrid`. This is done by setting the `DataSource` to the table in the `DataSet` of the passed object and calling the `BindData` method of the `DataGrid`. In addition, the `DataKeyField` is set to the field in the table that contains the primary key data. This provides the unique identifier for each row of test data in the rendered `DataGrid` and is needed to update the data submitted back to the server. The `bindData` method then persists the passed object to the `ViewState`. This causes the `testData` object to be serialized to a string and placed in the `ViewState` when the page is rendered.

When a page is rendered and sent to the client browser, all objects created in the code-behind are destroyed. This means the `testData` object no longer exists after the page is rendered unless you have persisted it. This is where data binding in web applications differs from data binding in Windows applications. In a Windows application, the underlying `DataSet`, `DataTable`, or other data container bound to the controls on the Windows form continues to exist and remains connected to the bound controls. Any changes made to the data in the bound controls are updated in the underlying data container, making updates to the original data source simple. In web forms, the connection to the underlying data container is broken when the page is rendered. Emulating the two-way binding and data updates available in Windows applications requires a bit of additional work, but it can be well worth the effort. (Again, our focus here assumes the bound control is a `DataGrid`. The `GridView` inherently supports this capability.)

When the user clicks the Update button, the form is submitted back to the server. ASP.NET takes care of updating the DataGrid server control with the data posted back to the server. However, the underlying data container (the testData object in this example) is not recreated by ASP.NET. Therefore, the testData object is rehydrated from the ViewState in the Update button click event.

After rehydrating the testData object, the contents of the object need to be updated with the current data in the DataGrid. This is done by iterating through the rows in the grid, extracting the individual data values, and updating the contents of the testData object.

For each row, the primary key value is obtained from the DataKeys collection for the current item. This is possible because the DataKeyField was set to the column containing the primary key when the DataGrid was originally bound.

Next, a reference is obtained to the first grade score text box. This is done by using the FindControl method of the current DataGrid item. This is possible because the text boxes defined in the DataGrid have been assigned IDs used here to perform the lookup. After the ID has been retrieved, the first grade score is updated in the testData object with the value in the text box. This process is repeated for the second grade score text box.

Production code should have validation to ensure that only numeric values are entered in the text box. This is best done using validation controls (see Chapter 3) or a custom control that allows only numeric entry, as described in Recipe 6.5.

After updating the contents of the object, the update method is called to update the contents of the database with the data in the object, and the data is again bound to the DataGrid.

The CH07TestDataVB class (see Examples 7-19 for the VB version and 7-20 for the C# version) provides the data services for this example. It encapsulates the data along with the methods for operating on the data. The class is defined with the Serializable attribute to provide the ability to serialize the data in the object created from the class to a string, which can then be stored in the ViewState or another location, such as a database. This string can then be deserialized to rehydrate the original object.

The class is designed to contain a DataSet as the container for the object data. This DataSet provides the ability to bind a DataGrid or another control directly to the data in the object. To bind to the internal data, two things are required. First, a property (the read-only testData property) must be provided to obtain a reference to the data table that contains the data.

Second, the names of the columns in the data table used for binding must be made available to bind controls to the specific data elements. These are defined as public constants in the class to provide a loose coupling between the code-behind and the data services class. By using the constants, the names of the columns can change without affecting any of the code that uses the CH07TestDataVB class.

In this example, the constructor for the class queries the database for the test data and fills the private member mTestData. The only special operation performed is to define the data column that is the primary key for the table in the DataSet. This makes it possible to find a specific row using the primary key value.

If the primary key is not defined for a table in the DataSet, the code will be required to filter the data in the table using a DataView or to iterate through all of the rows in the data table to find the row of interest. Both of these approaches can penalize performance.

Two properties are defined to set the first and second grade scores. Both of these properties perform the same actions but on different columns in the data table. To set a value, the appropriate row mus be located. This is done by using the Find method of the row collection with the testID (primary key) value passed to the property. The find operation will only work if a primary key is defined for the table. After finding the row matching the passed testID, the value is set to the passed score.

When all data has been changed, as required in the object, the Update method is called. This method updates the data from the object to the database. It utilizes the functionality in ADO.NET that will, with a few lines of code, perform all inserts, updates, and deletes required to match the data in the database with the data in the data table. This basically requires four steps:

1. Open a connection to the database. This is the same process used in the constructor.
2. Create a new data adapter using the same Select statement used to populate the data table originally.

   All of the columns currently contained in the data table must be included in the Select statement. For this reason, it is good to define the Select statement as a constant or a private member variable to allow the constructor and Update methods to use the same Select statement, as shown in the following code:

   ```
   da = New OleDbDataAdapter(CMD_TEXT, _
        dbConn)
   ```

   ```
   da = new OleDbDataAdapter(CMD_TEXT,
        dbConn);
   ```

3. Create a new command builder with a reference to the data adapter created in step 2. The command builder will build the appropriate insert, update, and delete commands from the provided Select command.
4. Call the Update method of the data adapter. This will perform all required inserts, updates, and deletes required to cause the data in the database to match the data in the data table.

> The update works because every row in a data table has a status property that indicates if the row in the data table has been modified, added, or deleted. From this status information, the data adapter can determine what actions are required.

This example does not provide the ability to add new tests or delete current tests. You can easily add the functionality to the `CH07TestDataVB` class by adding two methods: `addTest` and `deleteTest`. Your `addTest` method needs to be passed the test name, its maximum score, and the scores for first and second grade. With the passed data, it should add a new row to the data table containing the test data. Likewise, your `deleteTest` method should delete the required row from the data table using the `Delete` method of the rows collection. The `Delete` method does not delete the row but marks it for deletion when the update is performed.

> Though the technique described in this example is powerful and useful, you should use it carefully. Placing a serialized object in the `ViewState` can increase the size of the information sent to the client browser, and, because the `ViewState` information is stored in a hidden form field, it is sent back to the server when the page is submitted. In most cases, this doesn't matter, because an extra few thousand bytes is not a problem. However, if you have a large object, the overhead could be excessive. One way to use this technique with a large object is to store the object in the `Session` object instead of in the `ViewState`. The trade-off is the server resources and time required to transmit the data to the `Session` object. Refer to Recipe 7.2 for information on using the `Session` object.

See Also

Recipes 6.5, 7.2, and 16.6

Example 7-16. Maintaining page state using the ViewState (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH07ViewStateVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH07ViewStateVB"
 Title="ViewState Persistence" %>
<%@ Import Namespace="ASPNetCookbook.VBExamples" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Maintaining Page State With The ViewState (VB)
 </div>
 <table width="90%" align="center" border="0">
  <tr>
   <td align="center">
```

```
    <asp:DataGrid ID="dgScores" Runat="server"
      Width="100%"
      AutoGenerateColumns="False"
       BorderColor="000080"
      BorderWidth="2px"
       HeaderStyle-BackColor="#000080"
      HeaderStyle-CssClass="tableHeader"
      ItemStyle-CssClass="tableCellNormal"
      ItemStyle-BackColor="#FFFFFF" >
    <Columns>
    <asp:TemplateColumn HeaderStyle-HorizontalAlign="Center"
           HeaderText="Test Name">
    <ItemTemplate>
     <%#Eval(CH07TestDataVB.TEST_NAME)%>
    </ItemTemplate>
    </asp:TemplateColumn>
    <asp:TemplateColumn HeaderStyle-HorizontalAlign="Center"
           ItemStyle-HorizontalAlign="Center"
           HeaderText="Max Score">
    <ItemTemplate>
     <%#Eval(CH07TestDataVB.MAX_SCORE)%>
    </ItemTemplate>
    </asp:TemplateColumn>
    <asp:TemplateColumn HeaderStyle-HorizontalAlign="Center"
           ItemStyle-HorizontalAlign="Center"
           HeaderText="First Grade<br />Passing Score">
    <ItemTemplate>
     <asp:TextBox id="txtFirstGradeScore" runat="server"
      Columns="5"
      text='<%# Eval(CH07TestDataVB.FIRST_GRADE_PASSING_SCORE) %>' />
     </ItemTemplate>
    </asp:TemplateColumn>
    <asp:TemplateColumn HeaderStyle-HorizontalAlign="Center"
            ItemStyle-HorizontalAlign="Center"
            HeaderText="Second Grade<br />Passing Score">
    <ItemTemplate>
     <asp:TextBox id="txtSecondGradeScore" runat="server"
      Columns="5"
      text='<%# Eval(CH07TestDataVB.SECOND_GRADE_PASSING_SCORE) %>' />
    </ItemTemplate>
    </asp:TemplateColumn>
    </Columns>
    </asp:DataGrid>
   </td>
  </tr>
  <tr>
   <td align="center">
    <br />
    <asp:Button ID="btnUpdate" runat="server"
       Text="Update"
       OnClick="btnUpdate_Click" />
    <asp:Button ID="btnCancel" runat="server"
```

```
       Text="Cancel"
       OnClick="btnCancel_Click" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 7-17. Maintaining page state using the ViewState code-behind (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH07ViewStateVB.aspx
 ''' </summary>
 Partial Class CH07ViewStateVB
  Inherits System.Web.UI.Page
  'the following constant defines the name of the viewstate variable used
  'to store the test data object
  Private Const VS_TEST_DATA As String = "TestData"

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim testData As CH07TestDataVB

   If (Not Page.IsPostBack()) Then
    'create new test data object and bind to it
    testData = New CH07TestDataVB
    bindData(testData)
   End If
  End Sub 'Page_Load

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the update button click
  ''' event event. It is responsible for updating the contents of the
  ''' database with the data from the form.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnUpdate_Click(ByVal sender As Object, _
          ByVal e As System.EventArgs)

   'the following constants define the names of the textboxes in the
   'datagrid rows
   Const FIRST_GRADE_SCORE_TEXTBOX As String = "txtFirstGradeScore"
   Const SECOND_GRADE_SCORE_TEXTBOX As String = "txtSecondGradeScore"

   Dim testData As CH07TestDataVB
   Dim item As System.Web.UI.WebControls.DataGridItem
   Dim txtScore As TextBox
   Dim testID As Integer

   'make sure page contents are valid
   If (Page.IsValid) Then
    'rehydrate the test data object from the viewstate
    testData = CType(ViewState.Item(VS_TEST_DATA), _
      CH07TestDataVB)
    'copy the contents of the fields in the datagrid to the test data
    'object to emulate the two-way databinding in the Windows world
    For Each item In dgScores.Items
    'get the testID for the test data in the datagrid row
    testID = CInt(dgScores.DataKeys.Item(item.ItemIndex))

    'get a reference to the first grade score textbox in the row
    txtScore = CType(item.FindControl(FIRST_GRADE_SCORE_TEXTBOX), _
       TextBox)

    'update the first grade score in the test data object
    testData.firstGradeScore(testID) = CInt(txtScore.Text)

    'get a reference to the second grade score textbox in the row
    txtScore = CType(item.FindControl(SECOND_GRADE_SCORE_TEXTBOX), _
       TextBox)

    'update the first grade score in the test data object
    testData.secondGradeScore(testID) = CInt(txtScore.Text)
    Next item

    'update the test data in the database
    testData.update()

    'rebind the data to the datagrid
    bindData(testData)
   End If
  End Sub 'btnUpdate_Click
```

```
   '''*************************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the cancel button click
   ''' event event. It is responsible for cancel the current edits.
   ''' </summary>
   '''
   ''' <param name="sender">Set to the sender of the event</param>
   ''' <param name="e">Set to the event arguments</param>
   Protected Sub btnCancel_Click(ByVal sender As Object, _
         ByVal e As System.EventArgs)

    'perform the actions required to cancel the edits
   End Sub 'btnCancel_Click

   '''*************************************************************************
   ''' <summary>
   ''' This routine binds the data in the passed object to the datagrid then
   ''' persists the object in the viewstate
   ''' </summary>
   '''
   ''' <param name="testData">Set to the test data to bind to the datagrid
   ''' on the form
   ''' </param>
   Private Sub bindData(ByVal testData As CH07TestDataVB)
    'bind the test data to the datagrid
    dgScores.DataSource = testData.testData()
    dgScores.DataKeyField = testData.TEST_DATA_ID
    dgScores.DataBind()

    'save the test data object in the view state
    ViewState.Add(VS_TEST_DATA, _
       testData)
   End Sub 'bindData
  End Class 'CH07ViewStateVB
End Namespace
```

Example 7-18. Maintaining page state using the ViewState code-behind (.cs)

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

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
  ///  CH07ViewStateCS.aspx
/// </summary>
public partial class CH07ViewStateCS : System.Web.UI.Page
{
 // the following constant defines the name of the viewstate variable used
 // to store the test data object
 private const String VS_TEST_DATA = "TestData";

 ///***********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
  CH07TestDataCS testData = null;

  if (!Page.IsPostBack)
  {
   // create new test data object and bind to it
   testData = new CH07TestDataCS();
   bindData(testData);
  }
 } // Page_Load

 ///***********************************************************************
 /// <summary>
 /// This routine provides the event handler for the update button click
 /// event event. It is responsible for updating the contents of the
 /// database with the data from the form.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnUpdate_Click(object sender, EventArgs e)
 {
  // the following constants define the names of the textboxes in the
  // datagrid rows
  const String FIRST_GRADE_SCORE_TEXTBOX = "txtFirstGradeScore";
  const string SECOND_GRADE_SCORE_TEXTBOX = "txtSecondGradeScore";

  CH07TestDataCS testData = null;
  TextBox txtScore = null;
  int testID;

  // make sure page contents are valid
  if (Page.IsValid)
  {
   // rehydrate the test data object from the viewstate
   testData = (CH07TestDataCS)(ViewState[VS_TEST_DATA]);
```

```
    // copy the contents of the fields in the datagrid to the test data
    // object to emulate the two-way databinding in the Windows world
    foreach (DataGridItem item in dgScores.Items)
    {
    // get the testID for the test data in the datagrid row
    testID = Convert.ToInt32(dgScores.DataKeys[item.ItemIndex]);

    //get a reference to the first grade score textbox in the row
    txtScore = (TextBox)(item.FindControl(FIRST_GRADE_SCORE_TEXTBOX));

    // update the first grade score in the test data object
    testData.set_firstGradeScore(testID,
          Convert.ToInt32(txtScore.Text));
    // get a reference to the second grade score textbox in the row
    txtScore = (TextBox)(item.FindControl(SECOND_GRADE_SCORE_TEXTBOX));

    // update the first grade score in the test data object
    testData.set_secondGradeScore(testID,
           Convert.ToInt32(txtScore.Text));
    } // foreach

    // update the test data in the database
    testData.update();

    // rebind the data to the datagrid
    bindData(testData);
   } // if (Page.IsValid)
  } // btnUpdate_Click

  ///************************************************************************
  /// <summary>
  /// This routine provides the event handler for the cancel button click
  /// event event. It is responsible for cancel the current edits.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnCancel_Click(object sender, EventArgs e)
  {
   // perform the actions required to cancel the edits
  } // btnCancel_Click

  ///************************************************************************
  /// <summary>
  /// This routine binds the data in the passed object to the datagrid then
  /// persists the object in the viewstate
  /// </summary>
  ///
  /// <param name="testData">Set to the test data to bind to the datagrid
  /// on the form
  /// </param>
  private void bindData(CH07TestDataCS testData)
  {
   // bind the test data to the datagrid
   dgScores.DataSource = testData.testData;
    dgScores.DataKeyField = CH07TestDataCS.TEST_DATA_ID;
   dgScores.DataBind();

   // save the test data object in the view state
   ViewState.Add(VS_TEST_DATA,
      testData);
  } // bindData
 } // CH07ViewStateCS
}
```

Example 7-19. Data service class for storage in the ViewState (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Configuration.ConfigurationManager
Imports System.Data
Imports System.Data.OleDb


Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides an encapsulation of test data and properties/method
 ''' to operate on the data.
 '''
 ''' NOTE: This class is marked as serializable to provide the ability to
 '''    serialize the objects created with the class to an XML string.
 ''' </summary>
 <Serializable()> _
 Public Class CH07TestDataVB
  'constants used to bind the data in the encapsulated dataset
  Public Const TEST_DATA_ID As String = "TestDataID"
  Public Const TEST_NAME As String = "TestName"
  Public Const FIRST_GRADE_PASSING_SCORE As String = "FirstGradePassingScore"
  Public Const SECOND_GRADE_PASSING_SCORE As String = "SecondGradePassingScore"
  Public Const MAX_SCORE As String = "MaxScore"

  'constant to provide the name of the table in the dataset
  Private Const TEST_DATA_TABLE As String = "TestData"

  'private attributes
  Private mTestData As DataSet
  Private mConnectionStr As String

  'the following constant is used to query the data from the database
  Private Const CMD_TEXT As String = "SELECT " & TEST_DATA_ID & "," & _
          TEST_NAME & "," & _
          FIRST_GRADE_PASSING_SCORE & "," & _
          SECOND_GRADE_PASSING_SCORE & "," & _
          MAX_SCORE & _
          " FROM " & TEST_DATA_TABLE

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get a reference to the table
  ''' in the dataset containing the test data.
  ''' </summary>
  Public ReadOnly Property testData() As DataTable
   Get
    Return (mTestData.Tables(TEST_DATA_TABLE))
   End Get
  End Property 'testData

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the first grade score
  ''' for the passed test ID.
  ''' </summary>
  Public Property firstGradeScore(ByVal testID As Integer) As Integer
   Get
    Dim dRow As DataRow

    'get row with the passed testID value
    dRow = mTestData.Tables(TEST_DATA_TABLE).Rows.Find(testID)

    'return the first grade passing score
    Return (CInt(dRow.Item(FIRST_GRADE_PASSING_SCORE)))
   End Get
   Set(ByVal Value As Integer)
    Dim dRow As DataRow

    'get row with the passed testID value
    dRow = mTestData.Tables(TEST_DATA_TABLE).Rows.Find(testID)

    'set the first grade passing score
    dRow.Item(FIRST_GRADE_PASSING_SCORE) = Value
   End Set
  End Property 'firstGradeScore

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the second grade score
  ''' for the passed test ID.
  ''' </summary>
```

```
Public Property secondGradeScore(ByVal testID As Integer) As Integer
 Get
  Dim dRow As DataRow

  'get row with the passed testID value
  dRow = mTestData.Tables(TEST_DATA_TABLE).Rows.Find(testID)

  'return the first grade passing score
  Return CInt((dRow.Item(SECOND_GRADE_PASSING_SCORE)))

 End Get

 Set(ByVal Value As Integer)
  Dim dRow As DataRow

  'get row with the passed testID value
  dRow = mTestData.Tables(TEST_DATA_TABLE).Rows.Find(testID)

  'set the first grade passing score
  dRow.Item(SECOND_GRADE_PASSING_SCORE) = Value

 End Set
End Property 'secondGradeScore

'''*************************************************************************
''' <summary>
''' This routine provides the ability to update the test data for this
''' object in the database.
''' </summary>
Public Sub update()
 Dim dbConn As OleDbConnection = Nothing
 Dim da As OleDbDataAdapter = Nothing
 Dim cmdBuilder As OleDbCommandBuilder = Nothing

 Try
 dbConn = New OleDbConnection(mConnectionStr)
 dbConn.Open()
 da = New OleDbDataAdapter(CMD_TEXT, _
      dbConn)

 'create a command builder which will create the appropriate update,
 'insert, and delete SQL statements
 cmdBuilder = New OleDbCommandBuilder(da)

 'update data in the testdata table
 da.Update(mTestData, _
    TEST_DATA_TABLE)

Finally
 'cleanup
 If (Not IsNothing(dbConn)) Then
  dbConn.Close()
```

```
    End If
   End Try
  End Sub 'update

  '''**************************************************************************
  ''' <summary>
  ''' This constructor creates the object and populates it with test data
  ''' from the database
  ''' </summary>
  Public Sub New()
   Dim dbConn As OleDbConnection = Nothing
   Dim da As OleDbDataAdapter = Nothing
   Dim key(0) As DataColumn

   Try
    'get the connection string from web.config and open a connection
    'to the database
    mConnectionStr = _
     ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDbConnection(mConnectionStr)
    dbConn.Open()

    'get the data from the database
    da = New OleDbDataAdapter(CMD_TEXT, _
         dbConn)
    mTestData = New DataSet
    da.Fill(mTestData, _
     TEST_DATA_TABLE)

    'define the testID column in the data table as a primary key column
    'this makes it possible to "lookup" a datarow with the testID value
    'NOTE: The PrimaryKey property expects an array of DataColumn even
    '    when only one column is used as the primary key
    key(0) = mTestData.Tables(TEST_DATA_TABLE).Columns(TEST_DATA_ID)
    mTestData.Tables(TEST_DATA_TABLE).PrimaryKey = key
    Finally
     'cleanup
     If (Not IsNothing(dbConn)) Then
     dbConn.Close()
     End If
    End Try
   End Sub 'New
  End Class 'CH07TestDataVB
 End Namespace
```

Example 7-20. Data service class for storage in the ViewState (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides an encapsulation of test data and properties/method
 /// to operate on the data.
 ///
 /// NOTE: This class is marked as serializable to provide the ability to
 ///    serialize the objects created with the class to an XML string.
 /// </summary>
 [Serializable]
 public class CH07TestDataCS
 {
  // constants used to bind the data in the encapsulated dataset
  public const String TEST_DATA_ID = "TestDataID";
  public const String TEST_NAME = "TestName";
  public const String FIRST_GRADE_PASSING_SCORE = "FirstGradePassingScore";
  public const String SECOND_GRADE_PASSING_SCORE = "SecondGradePassingScore";
  public const String MAX_SCORE = "MaxScore";

  // constant to provide the name of the table in the dataset
  private const String TEST_DATA_TABLE = "TestData";

  // private attributes
  private DataSet mTestData = null;
  private String mConnectionStr = null;

  // the following constant is used to query the data from the database
  private const String CMD_TEXT = "SELECT " + TEST_DATA_ID + "," +
       TEST_NAME + "," +
       FIRST_GRADE_PASSING_SCORE + "," +
       SECOND_GRADE_PASSING_SCORE + "," +
       MAX_SCORE +
       " FROM " + TEST_DATA_TABLE;

  ///***************************************************************************
  /// <summary>
  /// This property provides the ability to get a reference to the table
  /// in the dataset containing the test data.
  /// </summary>
  public DataTable testData
  {
   get
   {
    return (mTestData.Tables[TEST_DATA_TABLE]);
   }
  } // testData
   ///***********************************************************************
  /// <summary>
  /// This routine provides the ability to set the first grade score for
  /// the passed test ID.
  /// </summary>
  ///
  /// <param name="testID">Set to the ID of the test</param>
  /// <param name="score">Set to the test score</param>
  public void set_firstGradeScore(int testID, int score)
  {
   DataRow dRow = null;

   // get row with the passed testID value
   dRow = mTestData.Tables[TEST_DATA_TABLE].Rows.Find(testID);

   // set the first grade passing score
   dRow[FIRST_GRADE_PASSING_SCORE] = score;
  } // set_firstGradeScore

   ///***********************************************************************
  /// <summary>
  /// This routine provides the ability to set the second grade score for
  /// the passed test ID.
  /// </summary>
  ///
  /// <param name="testID">Set to the ID of the test</param>
  /// <param name="score">Set to the test score</param>
  public void set_secondGradeScore(int testID, int score)
  {
   DataRow dRow = null;

   // get row with the passed testID value
   dRow = mTestData.Tables[TEST_DATA_TABLE].Rows.Find(testID);

   // set the second grade passing score
   dRow[SECOND_GRADE_PASSING_SCORE] = score;
  } // set_secondGradeScore

   ///***********************************************************************
  /// <summary>
  /// This routine provides the ability to update the test data for this
  /// object in the database.
  /// </summary>
  public void update()
  {
   OleDbConnection dbConn = null;
   OleDbDataAdapter da = null;
   OleDbCommandBuilder cmdBuilder = null;

   try
   {
    dbConn = new OleDbConnection(mConnectionStr);
```

```
    dbConn.Open();

    da = new OleDbDataAdapter(CMD_TEXT, dbConn);

    // create a command builder which will create the appropriate update,
    // insert, and delete SQL statements
    cmdBuilder = new OleDbCommandBuilder(da);

    // update data in the testdata table
    da.Update(mTestData,
      TEST_DATA_TABLE);
   }

   finally
   {
    // cleanup
    if (dbConn != null)
    {
    dbConn.Close();
    }
   } // finally

  } // update

  ///*************************************************************************
  /// <summary>
  /// This constructor creates the object and populates it with test data
  /// from the database
  /// </summary>
  public CH07TestDataCS()
  {
   OleDbConnection dbConn = null;
   OleDbDataAdapter da = null;
   DataColumn[] key = new DataColumn[1];

   try
   {
    // get the connection string from web.config and open a connection
    // to the database
    mConnectionStr = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(mConnectionStr);
    dbConn.Open();

    // get the data from the database
    da = new OleDbDataAdapter(CMD_TEXT,
         dbConn);
    mTestData = new DataSet();
    da.Fill(mTestData,
    TEST_DATA_TABLE);

    // define the testID column in the data table as a primary key column
```

```
     // this makes it possible to "lookup" a datarow with the testID value
     // NOTE: The PrimaryKey property expects an array of DataColumn even
     //   when only one column is used as the primary key
     key[0] = mTestData.Tables[TEST_DATA_TABLE].Columns[TEST_DATA_ID];
     mTestData.Tables[TEST_DATA_TABLE].PrimaryKey = key;
    } // try

    finally
    {
     // cleanup
     if (dbConn != null)
     {
     dbConn.Close();
     }
    } // finally
   } // CH07TestDataCS
  } // CH07TestDataCS
 }
```

PREV

Chapter 8. Error Handling

PREV

8.0 Introduction

Journeyman programmers know that proper error handling is critical to the operation of an application. Without it, your chances of making the application fault tolerant are less than remote. Taking the time to plan an error-handling strategy in the early stages of a project can pay off as the project progresses. Yet, too often error handling is given short shrift, for want of time, interest, awareness, accessibility, and who knows what.

Fortunately, error handling has been greatly improved in ASP.NET, making it more approachable and easier to implement than classic ASP. Taking a page or two from the Java playbook, ASP.NET provides state-of-the-art handling of errors with exceptions and error-handler events.

The error-handling model in ASP.NET lets you handle errors easily at the method, page, and application levels of your web applications. Most applications will use some combination of these to handle problems when they arise. In this chapter, we have included recipes for handling errors at each level:

Method level

When does it make sense to handle errors locally versus letting them propagate to a higher level? In general, you want to handle recoverable errors in the method where they occur and let nonrecoverable errors propagate up. Recipe 8.1 details this subject. It helps you understan all the nuances of the `try…Catch…Finally` block and includes sets of `if…then` solution statements and leading questions to help you choose how to implement error handling in a routine.

Page level

Recipe 8.2 shows you how to trap errors in a page and redirect the user to another page. Why would you want to use this approach? It allows you to handle allpage-level errors in a uniform way, which can simplify error-handling code and make it more consistent and robust. The trick is in keeping all the error-handling code in one place in the original page by leveraging the `Page_Error` method, as the recipe explains.

Application level

Recipe 8.3 shows you how to handle the errors at the application level that occur on any page of your application. The approach we advocate involves trapping errors that occur and logging them in an event log prior to redirecting the user to another page. By handling all exceptions a the application level, rather than at the method or page level, you can process all errors for the application in a single location. Doing allerror handling in one place in an application is key to writing effective code. It requires understanding what happens to unhandled exceptions at the method, page, and application levels, which this recipe explains.

The final recipe is about creating user-friendly error messages, which sounds simple but involves creating a new exception class that inherits from the .NET Framework's base exception classes and adding the functionality your application requires. How to take advantage of this new class in the `Catch` block of your code is also explained.

PREV

PREV

Recipe 8.2. Handling Errors at the Method Level

Problem

You're uncertain how to organize your code to handle errors at the method level. In particular, you'd like to take advantage of .NET structured exception handling for dealing with errors, but you're unsure how to best implement it.

Solution

If potential errors are recoverable in the routine

Use a combination of `try…Catch` blocks as a retry mechanism for error handling.

If useful information can be added to the exception

Create and throw a new exception with the added information.

If cleanup is required

Perform it in the `finally` block.

If potential errors are not recoverable in the routine

Recovery should be handled by the calling routine and its error-handling structure.

Discussion

Because .NET structured exception handling is so good, we recommend you use it, or consider using it, with every method that you write. There are a number of ways to implement its functionality.

Basic syntax of Try…Catch…Finally

To begin with, here is the syntax of a .NET `try…Catch…Finally` block in VB and C#:

VB

```
 Private Sub anyRoutine( )
  Try
  'Routine code in this block

  Catch exc As Exception
  'error handling in this block

  Finally
  'cleanup performed in this block

  End Try
  End Sub 'anyRoutine
```

C#

```
  private void anyRoutine( )
  {
   try
   {
    // Routine code in this block
   }

   catch (Exception exc)
   {
    // error handling in this block
   }

   finally
   {
    // cleanup performed in this block
   }
  } // anyRoutine
```

The `try` block includes code that implements the method.

The `catch` block, which is optional, includes code to handle specific errors that you have identified as likely to occur and to recover from them when that is possible.

The `finally` block code, which is optional, performs any cleanup required on leaving the method, whether due to an error or not. This typically includes the closing of any open database connections and files, disposing of objects created by the method, and so on. A`finally` block is guaranteed to be executed even if an exception is thrown or the code in the routine performs a return.

As noted, the `catch` and `finally` blocks are optional. Sometimes, you'll want to use one or the other, and other times you'll want to use both.

A try block must contain either a catch or a finally block.

Guidelines for implementing

Developers should make use of .NET exception handling in any method where an error is possible, but the exact technique depends on the circumstances, as summarized in Table 8-1.

Table 8-1. Guidelines for Try...Catch...Finally blocks

Can errors occur?	Recoverable?	Can useful context information be added?	Cleanup required?	Recommended combination of try, catch, and finally
No	N/A	N/A	No	None
No	N/A	N/A	Yes	TRy and finally only
Yes	No	No	No	None
Yes	No	No	Yes	TRy and finally only
Yes	No	Yes	No	try and catch only
Yes	No	Yes	Yes	try, catch, and finally
Yes	Yes	N/A	N/A	try and catch only

The .NET Framework does not close database connections, files, etc., when an error occurs. This is your responsibility as a programmer, and you should do it in the finally block. The finally block is the last opportunity to perform any cleanup before the exception-handling infrastructure takes control of the application.

Additional considerations

To help you implement error handling in a routine, we've provided the following leading questions. Your answers can help you determine what portions of the try…Catch…Finally block you need. Refer to Table 8-1 for how to structure a routine based on your answers.

Can any errors occur in this routine?

If not, no special error-handling code is required. Do not shortchange the answer to this question, however, because even `x=x+1` can result in an overflow exception.

Are the potential errors recoverable in the routine?

If an error occured but nothing useful can be done in the routine, the exception should be allowed to propagate to the calling routine. It serves no useful purpose to catch the exception and rethrow it. This question is different from, "Are the potential errors recoverable at the application level?" For example, if the routine attempts to write a record to a database and finds the record locked, a retry can be attempted in the routine. However, if a value is passed to the routine and the operations on the value result in an overflow or other error, recovery cannot be performed in the routine but should be handled by the calling routine and its error-handling structure.

Can any useful information be added to the exception?

Exceptions that occur in the .NET Framework contain detailed information regarding the error. However, the exceptions do not provide any context information about what was being attempted at the application level that may assist in troubleshooting the error or providing more useful information to the user.

A new exception can be created and thrown with the added information. The first parameter fo the new exception object should contain the useful context message, and the second parameter should be the original exception. The exception-handling mechanisms in the .NET Framework create a linked list of `Exception` objects to create a trail from the root of the exception up to the level where the exception is handled. By passing the original exception as the second parameter, the linked list from the root exception is maintained, as in this example

```
Catch exc As Exception
 Throw New Exception("Useful context message", _
   exc)
```

```
catch (Exception exc)
{
 throw (new Exception("Useful context message",
   exc));
}
```

Is a combination of `Try…Catch` blocks warranted?

A combination of `TRy…Catch` blocks can help provide a retry mechanism for error handling, as shown in Examples 8-1 (VB) and 8-2 (C#). These examples show the use of an internal `try…Catch` block within a `while` loop to provide a retry mechanism and an overall `TRy…Finally` block to ensure cleanup is performed.

The catch block should not be used for normal program flow. Normal program flow code should only be placed in theTRy block, with the abnormal flow being placed in the catch block. Using the catch block in normal program flow will result in significant performance degradation due to the complex operations being performed by the .NET Framework to process exceptions.

The exception-handling mechanisms in the .NET Framework are powerful. Whereas this example highlights exception handling at the method level, other specific exception types can be caught and processed differently. In addition, you can create new exception classes by inheriting from the base exception classes and adding the functionality required by your applications. An example of this technique is shown in Recipe 8.4.

See Also

Recipe 8.4

Example 8-1. Retrying when an exception occurs (.vb)

```
Imports System
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

 …

 Private Sub updateData(ByVal problemID As Integer, _
       ByVal sectionHeading As String)
  Const MAX_RETRIES As Integer = 5

  Dim dbConn As OleDbConnection = Nothing
  Dim dCmd As OleDbCommand = Nothing
  Dim strConnection As String
  Dim cmdText As String
  Dim updateOK As Boolean
  Dim retryCount As Integer

  Try
   'get the connection string from web.config and open a connection
   'to the database
   strConnection = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
   dbConn = New OleDbConnection(strConnection)
   dbConn.Open( )

   'build the update SQL to update the record in the database
```

```
    cmdText = "UPDATE EditProblem " &_
       "SET SectionHeading='" & sectionHeading & "' " &_
       "WHERE EditProblemID=" &problemID.ToString( )
    dCmd = New OleDbCommand(cmdText, _
       dbConn)

    'provide a loop with a try catch block to facilitate retrying
    'the database update
    updateOK = False
    retryCount = 0
    While ((Not updateOK) And (retryCount < MAX_RETRIES))
     Try
     dCmd.ExecuteNonQuery( )
     updateOK = True

     Catch exc As Exception
     retryCount += 1
     If (retryCount >= MAX_RETRIES) Then
     'throw a new exception with a context message stating that
     'the maximum retries was exceeded
     Throw New Exception("Maximum retries exceeded", _
        exc)

     End If

    End Try
   End While

  Finally
   'cleanup
   If (Not IsNothing(dbConn)) Then
    dbConn.Close( )
   End If
  End Try
 End Sub 'updateData
```

Example 8-2. Retrying when an exception occurs (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;

 private void updateData(int problemID,
    String sectionHeading)
 {
```

```
const int MAX_RETRIES = 5;

OleDbConnection dbConn = null;
OleDbCommand dCmd = null;
String strConnection = null;
String cmdText = null;
bool updateOK;
int retryCount;

try
{
 // get the connection string from web.config and open a connection
 // to the database
 strConnection = ConfigurationManager.
  ConnectionStrings["dbConnectionString"].ConnectionString;
 dbConn = new OleDbConnection(strConnection);
 dbConn.Open( );

 // build the update SQL to update the record in the database
 cmdText = "UPDATE EditProblem " +
    "SET SectionHeading='" + sectionHeading + "' " +
    "WHERE EditProblemID=" + problemID.ToString( );
 dCmd = new OleDbCommand(cmdText, dbConn);
 // provide a loop with a try catch block to facilitate retrying
 // the database update
 updateOK = false;
 retryCount = 0;
 while ((!updateOK) &(retryCount < MAX_RETRIES))
 {
  try
  {
  dCmd.ExecuteNonQuery( );
  updateOK = true;
  } // try

  catch (Exception exc)
  {
  retryCount++;
  if (retryCount >= MAX_RETRIES)
  {
  // throw a new exception with a context message stating that
  // the maximum retries was exceeded
  throw new Exception("Maximum retries exceeded",
      exc);
  }
  } // catch
 } // While
} // try

finally
{
 // cleanup
```

```
    if (dbConn != null)
    {
     dbConn.Close( );
    }
   } // finally
  } // updateData
```

PREV

Recipe 8.3. Handling Errors at the Page Level

Problem

You want to trap any error that occurs on a page and, using a page-level event handler, redirect the user to another page that displays the information about the problem.

Solution

Add code to the `Page_Error` event handler of the page to set the `ErrorPage` property of that page to the URL you want to display to the user when an error occurs.

In the code-behind for the page, use the .NET language of your choice to:

1. Add a `Page_Error` event handler.
2. In the event handler, get a reference to the last error that occurred using the `GetLastError` method.
3. Set the `ErrorPage` property of the `Page` object to the URL of the page you want displayed after the error, adding `querystring` parameters to pass error information to the page.

Examples 8-3 (VB) and 8-4 (C#) demonstrate this solution. (Because the *.aspx* file for this example contains nothing related to the error handling, it is not included here.)

Discussion

The `Page_Error` event of the ASP.NET `Page` object is raised any time an unhandled error occurs in a page. The first action required in the event handler is to get a reference to the last error. After gettin the reference, the code should perform the required logging, notifications, etc. See Recipe 8.3 for an example of writing to the event log.

ASP.NET provides you with the ability to redirect the user to another page when an error occurs. To use this feature, set the `ErrorPage` property of the `Page` object to the URL of the page you want the user to see. You can add `querystring` parameters to the URL to pass specific error messages to the page. For instance, in the code snippets shown next, we've added three `querystring` parameters to the URL of an error message page: `PageHeader, Message1`, and `Message2`. `PageHeader` is set to the message "Error Occurred." `Message1` is set to the message in the `lastError` exception. This will be the message from the last exception thrown. `Message2` is a message we've added to say where the error was processed.

```
VB
 Page.ErrorPage = "CH08DisplayErrorVB.aspx" &_
     "?PageHeader=Error Occurred" &_
      "&Message1=" & lastError.Message &_
     "&Message2=" &_
     "This error was processed at the page level"
```

```
C#
 Page.ErrorPage = "CH08DisplayErrorCS.aspx" +
     "?PageHeader=Error Occurred" +
      "&Message1=" + lastError.Message +
     "&Message2=" +
     "This error was processed at the page level";
```

When the `Page_Error` event is completed, ASP.NET will automatically perform a redirect to the URL named in the `ErrorPage` property. You could do this yourself using `Response.Redirect`([*Page URL*]), but why write a line of code when ASP.NET can do it for you?

If you do not add any `querystring` parameters, ASP.NET will append one for you. The name of the parameter will be `aspxerrorpath`, and the value will be the relative URL to the specified page. In our example, the ASP.NET-added `querystring` would have been as follows:

```
aspxerrorpath=/ASPNetCookbook2VB/CH08PageLevelErrorHandlingVB.aspx
```

This technique, when coupled with those for handling exceptions at the method level (described in Recipe 8.1) can simplify handling errors in pages. The only place any code is required to gather error information and redirect to another page is in the `Page_Error` event handler. This is a significant improvement over the error-handling code required in classic ASP, where exception handling could b done only at the method level.

By default, ASP.NET displays the full error context in a special ASP.NET page on the local machine. If you access this example from a browser on the web server, the redirection described here will not occur. If you access this example from a different machine, the redirection will be performed. Refer to Chapter 12 when changing the default handling of error messages in *web.config*.

See Also

Recipes 8.1 and 8.3, Chapter 12

Example 8-3. Handling errors at the page level (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH08PageLevelErrorHandlingVB.aspx
 ''' </summary>
 Partial  Class  CH08PageLevelErrorHandlingVB
  Inherits System.Web.UI.Page
   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim values As Hashtable = Nothing

   'add a key/value pair to the hashtable without first creating
   'the hashtable which will cause a null reference exception error
   values.Add("Key", "Value")
  End Sub 'Page_Load

   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page  error event.
  ''' It builds a URL with the error information then sets the ErrorPage
  ''' property to the URL.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Page_Error(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Error
   Dim lastError As Exception

   'get the last error that occurred
   lastError = Server.GetLastError( )

   'do any logging, notifications, etc. here

   'set the URL of the page that will display the error and
   'include querystring parameters to allow the page to display
   'what happened
   Page.ErrorPage = "CH08DisplayErrorVB.aspx" &_
     "?PageHeader=Error Occurred" &_
     "&Message1=" &lastError.Message &_
     "&Message2=" &_
     "This error was processed at the page level"

  End Sub 'Page_Error
 End Class 'CH08PageLevelErrorHandlingVB
End Namespace
```

Example 8-4. Handling errors at the page level (.cs)

```
using System;
using System.Collections;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH08PageLevelErrorHandlingCS.aspx
 /// </summary>
 public partial class CH08PageLevelErrorHandlingCS : System.Web.UI.Page
 {
  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   Hashtable values = null;

   // add a key/value pair to the hashtable without first creating
   // the hashtable which will cause a null reference exception error
   values.Add("Key", "Value");
  } // Page_Load

  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page error event.
  /// It builds a URL with the error information then sets the ErrorPage
  /// property to the URL.
  /// </summary>
  ///
```

```
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   private void Page_Error(Object sender,
      System.EventArgs e)
   {
    Exception lastError = null;
    // get the last error that occurred
    lastError = Server.GetLastError( );

    // do any logging, notifications, etc. here

    // set the URL of the page that will display the error and
    // include querystring parameters to allow the page to display
    // what happened
    Page.ErrorPage = "CH08DisplayErrorCS.aspx" +
      "?PageHeader=Error Occurred" +
      "&Message1=" + lastError.Message +
      "&Message2=" +
      "This error was processed at the page level";
    } // Page_Error
  } //  CH08PageLevelErrorHandlingCS
}
```

Recipe 8.4. Handling Errors at the Application Level

Problem

You want to report and log all errors in a common location, regardless of where they arise within the application.

Solution

Incorporate the error handling in methods (described in Recipe 8.1), add code to the `Page_Error` event handler to rethrow the page errors, and add the code to the `Application_Error` event handler to perform the logging and redirection.

In the code-behind class for your ASP.NET pages that need to perform error handling, use the .NET language of your choice to:

1. Create a `Page_Error` event handler.
2. Rethrow the page errors from within the method. (This is needed to avoid all errors being wrapped with an `HttpUnhandledException` exception.)

In *global.asax*, use the .NET language of your choice to:

1. Create an `Application_Error` event handler.
2. Create a detailed message and write it to the event log.
3. Redirect the user to the error page using `Server.Transfer`.

The code we've written to demonstrate this solution is shown in Examples 8-6, 8-7, 8-8 through 8-9. The `Page_Error` code required in all pages is shown in Examples 8-6 (VB) and 8-7 (C#). The `Application_Error` code required in *global.asax* is shown in Examples 8-8 (VB) and 8-9 (C#). (Because the *.aspx* file for this example contains nothing related to the error handling, it is not included here.)

Discussion

The exception model in ASP.NET provides the ability for exceptions to be handled at any level, from the method level to the application level. An unhandled exception is sequentially rethrown to each method in the call stack. If no methods in the call stack handle the exception, the Page_Error event will be raised. If the exception is not handled in the Page_Error event, the event will be rethrown and the Application_Error event is raised. The rethrowing of exceptions to the application level allows fo processing in a single location for the application.

To process errors at the application level, each page must include the Page_Error event handler with a single line of code to rethrow the last exception that occurred:

VB

```
Private Sub Page_Error(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles MyBase.Error

  'rethrow the last error that occurred
 Throw Server.GetLastError( )
End Sub 'Page_Error
```

C#

```
private void Page_Error(Object sender,
   System.EventArgs e)
{
 // rethrow the last error that occurred
 throw Server.GetLastError( );
} // Page_Error
```

We do this step to avoid having the exception information wrapped with an HttpUnhandledException exception. ASP.NET automatically creates a new HttpUnhandledException at the page level unless you rethrow the last exception, which from ASP.NET's perspective constitutes handling the exception.

One school of thought says this step is unnecessary and that it's fine to have ASP.NET wrap your exceptions at will; all you have to do is ignore all the "outer" exceptions and get the first inner exception. We don't subscribe to this view, however, because there are cases, such as page parse errors, that do not get wrapped with the HttpUnhandledException. This can make it difficult to extract the "real" exception information when no guarantee exists that the "real" exception information is the first inner exception in the chain of exceptions.

Visual Studio users can make the insertion of the Page_Error code on each page much easier by using the built-in macro facilities or adding the code block for the Page_Error event to the toolbox. Alternatively, you can create a base page that contains the Page_Error method and have all of your pages inherit from this base page. With this approach, you do not have to deal with implementing Page_Error in all of your pages.

The error processing for the application is placed in the Application_Error event handler (in *global.asax*). Much of this code follows a standard pattern, which is illustrated in Examples 8-8 (VB) and 8-9 (C#). The first step is to get a reference to the last exception that occurred:

```
lastException = Server.GetLastError()
```

```
lastException = Server.GetLastError();
```

The next step is to create a detailed message to insert into the event log. This message should contain the message from the most recent exception and a complete dump of all error information in the link list of exceptions. The complete dump is obtained by calling the `ToString` method of the last exception, as in the following example code:

```
message = lastException.Message & _
    vbCrLf & vbCrLf & _
    lastException.ToString()
```

```
message = lastException.Message +
    "\r\r" +
    lastException.ToString();
```

Next, you can write the message to the event log. As we show in Examples 8-8 and 8-9, this is done by creating a new `EventLog` object, setting the `Source` property to a constant containing the name of the event source to write the information to (the Application log in our example), and writing the message to the event log. When writing the entry to the event log, the event type can be set to `Error`, `FailureAudit`, `Information`, `SuccessAudit`, and `Warning`, all of which are members of the `EventLogEntryType` enumeration. Here is the code responsible for writing to the event log in our example:

```
Log = New EventLog
Log.Source = EVENT_LOG_NAME
Log.WriteEntry(message, _
   EventLogEntryType.Error)
```

```
log = new EventLog( );
log.Source = EVENT_LOG_NAME;
log.WriteEntry(message,
   EventLogEntryType.Error);
```

The event log entry created by our example is shown in Example 8-5. The entry shows that a `NullReferenceException` occurred at line 41 in the code-behind for the example page. If the exceptio had been wrapped by throwing new exceptions at each error-handling point in the code, they would all have been listed here. This is useful for troubleshooting runtime errors because the source of the error is shown, along with the complete call path to the point the error occurred.

At this point any other notifications, such as sending an email to the system administrator, should be performed. Refer to Recipe 21.7 for information regarding sending emails.

The final step to processing errors at the application level is to clear the error and redirect the user t the page where an error message is displayed:

VB

```
 Server.ClearError( )
 Server.Transfer("CH08DisplayErrorVB.aspx"  &_
    "?PageHeader=Error Occurred" &_
     "&Message1=" &lastException.Message &_
     "&Message2=" &_
    "This error was processed at the application level")
```

C#

```
 Server.ClearError( );
 Server.Transfer("CH08DisplayErrorCS.aspx"  +
    "?PageHeader=Error Occurred" +
     "&Message1=" + lastException.Message +
    "&Message2=" +
    "This error was processed at the application level");
```

If you do not clear the error, ASP.NET will assume the error has not been processed and will handle it for you with its infamous "yellow" screen.

You can use one of two methods to redirect the user to the error page. The first method is to call `Response.Redirect`, which works by returning information to the browser instructing the browser to do a redirect to the page indicated. This results in an additional round trip to the server. As we show in Examples 8-8 (VB) and 8-9 (C#), the second method is `Server.Transfer`, which is the method we favor because it transfers the request to the indicated page without the extra browser/server round trip.

Calling `Response.Redirect` throws a `ThreadAbortException` as a result of aborting the execution of the page. This exception is handled differently than other exceptions and will not bubble up to the page or application level.

By default, Windows 2000 and 2003 Server provide three event sources: Application, Security, and System. Of the three sources, only the default ASP.NET user (ASPNET) has permission to write to the Application log. Attempts to write to the Security or System logs will result in an exception being thrown in the `Application_Error` event.

You can make the errors for your application easier to find in the event viewer by creating an event source specific to your application. Without escalating the privileges for the ASP.NET user to the System level (a bad option), you cannot create a new event source within your ASP.NET application. Two options are available. You can use the registry editor and add a new key to the `HKEY_LOCAL_MACHINE\System\ CurrentControlSet\Services\EventLog` key or create a console application to do the work for you. We suggest the console application because it is easy and repeatable.

Create a console application in the usual fashion, add the following code to it, and run the applicatio while being logged in as a user with administrative privileges. This will create an event source specifi to your application. The only change required to the code described here is to change the `EVENT_LOG_NAME` constant value to the name of your new event source.

```
 Const EVENT_LOG_NAME As String = "Your Application"

 If (Not EventLog.SourceExists(EVENT_LOG_NAME)) Then
  EventLog.CreateEventSource(EVENT_LOG_NAME, EVENT_LOG_NAME)
 End If
```

```
 const String EVENT_LOG_NAME = "Your Application";

 if (EventLog.SourceExists(EVENT_LOG_NAME) != null)
 {
  EventLog.CreateEventSource(EVENT_LOG_NAME, EVENT_LOG_NAME);
 }
```

See Also

Recipes 8.1 and 21.7

Example 8-5. Event log entry for this example

```
System.NullReferenceException: {"Object reference not set to an instance
 of an object."}
Data: {System.Collections.ListDictionaryInternal}
HelpLink: Nothing
HResult:  -2147467261
InnerException: Nothing
IsTransient: False
Message: "Object reference not set to an instance of an object."
Source:  "App_Web_3q7je6zg"
StackTrace: " at ASPNetCookbook.VBExamples.
 CH08ApplicationLevelErrorHandlingVB.Page_Error(Object  sender,  EventArgs  e)
 in  E:\ASPNetCookbook2\projects\ASPNetCookbookVB2\
 CH08ApplicationLevelErrorHandlingVB.aspx.vb:line  41
at System.Web.UI.TemplateControl.OnError(EventArgs e)
at System.Web.UI.Page.HandleError(Exception e)
at System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint,
 Boolean includeStagesAfterAsyncPoint)
at System.Web.UI.Page.ProcessRequest(Boolean includeStagesBeforeAsyncPoint,
 Boolean includeStagesAfterAsyncPoint)
at System.Web.UI.Page.ProcessRequest( )
at System.Web.UI.Page.ProcessRequest(HttpContext context)
at System.Web.HttpApplication.CallHandlerExecutionStep.System.Web.
 HttpApplication.IExecutionStep.Execute( )
at System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean&
 completedSynchronously)"
TargetSite: {System.Reflection.RuntimeMethodInfo}
```

Example 8-6. Page_Error code for handling errors at the application level (.vb)

```
'''***********************************************************************
''' <summary>
''' This routine handles the error event for the page. It is used to
''' trap all errors for the page and rethrow the exception. The rethrow
''' is needed to avoid all errors being wrapped with an
''' "HttpUnhandledException" exception.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub Page_Error(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Error
 'rethrow the last error that occurred
 Throw Server.GetLastError()
End Sub 'Page_Error
```

Example 8-7. Page_Error code for handling errors at the application level (.cs)

```
///***********************************************************************
/// <summary>
/// This routine handles the error event for the page. It is used to
/// trap all errors for the page and rethrow the exception. The rethrow
/// is needed to avoid all errors being wrapped with an
/// "HttpUnhandledException" exception.
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
private void Page_Error(Object sender,
    System.EventArgs e)
{
 // rethrow the last error that occurred
 throw Server.GetLastError( );
} // Page_Error
```

Example 8-8. Application_Error code for handling errors at the applicatior level (.vb)

```
'''***********************************************************************
''' <summary>
''' This routine provides the event handler for the application error
''' event. It is responsible for processing errors at the application
''' level.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
 Const EVENT_LOG_NAME As String = "Application"

 Dim lastException As Exception
 Dim Log As EventLog
 Dim message As String

 'get the last error that occurred
 lastException = Server.GetLastError()
 'create the error message from the message in the last exception along
 'with a complete dump of all of the inner exceptions (all exception
 'data in the linked list of exceptions)
 message = lastException.Message & _
     vbCrLf & vbCrLf & _
     lastException.ToString()

 'Insert error information into the event log
 Log = New EventLog
 Log.Source = EVENT_LOG_NAME
 Log.WriteEntry(message, _
    EventLogEntryType.Error)

 'perform other notifications, etc. here


 'clear the error and redirect to the page used to display the
 'error information
 Server.ClearError()
 Server.Transfer("CH08DisplayErrorVB.aspx"  &  _
    "?PageHeader=Error Occurred" & _
     "&Message1=" & lastException.Message & _
     "&Message2=" &_
    "This error was processed at the application level")
End Sub 'Application_Error
```

Example 8-9. Application_Error code for handling errors at the applicatior level (.cs)

```
///**********************************************************************
/// <summary>
/// This routine provides the event handler for the application error
/// event. It is responsible for processing errors at the application
/// level.
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
 protected void Application_Error(Object sender, EventArgs e)
 {
  const String EVENT_LOG_NAME = "Application";

  Exception lastException = null;
  System.Diagnostics.EventLog log = null;
  String message = null;

  // get the last error that occurred
  lastException = Server.GetLastError( );

  // create the error message from the message in the last exception along
  // with a complete dump of all of the inner exceptions (all exception
  // data in the linked list of exceptions)
  message = lastException.Message +
      "\r\r" +
      lastException.ToString( );

  // Insert error information into the event log
  log = new System.Diagnostics.EventLog( );
  log.Source = EVENT_LOG_NAME;
  log.WriteEntry(message,
    EventLogEntryType.Error);

  // perform other notifications, etc. here


  // clear the error and redirect to the page used to display the
  // error information
  Server.ClearError( );
  Server.Transfer("CH08DisplayErrorCS.aspx"  +
    "?PageHeader=Error Occurred" +
    "&Message1=" + lastException.Message +
    "&Message2=" +
    "This error was processed at the application level");
} // Application_Error
```

Recipe 8.5. Displaying User-Friendly Error Messages

Problem

You want the event-handling methods described in this chapter to write detailed messages to an error log for use in debugging your application, but you want to display friendly, informative messages to the user.

Solution

Create a custom exception class that includes a property to hold the user-friendly message, and when an error occurs, instantiate a new exception object of the custom type in the `Catch` block of your error-handling code, set the property of the exception to the desired message, and throw the new exception.

Use the .NET language of your choice to create the custom exception class by deriving from `System.ApplicationException` and adding a property to hold the userfriendly message, giving it a name like `userFriendlyMessage`.

In the code-behind for the ASP.NET pages of your application that need to perform error handling, use the .NET language of your choice to:

1. In the `Catch` block of methods where informative messages are useful, instantiate a new exception object of your custom class type, set the `userFriendlyMessage` property to the desired message, and throw the new exception.

2. In the `Application_Error` event handler, write the detailed information provided by the exception object to the event log and display the message contained in the `userFriendlyMessage` property of the exception on a common error message page.

The custom exception class we've created to demonstrate this solution is shown in Examples 8-10 (VB) and 8-11 (C#). The code showing how to create the new exception is shown in Examples 8-12 (VB) and 8-13 (C#). The code for the `Application_Error` event handler is shown in Examples 8-14 (VB) and 8-15 (C#). (Because the *.aspx* file for this example contains nothing related to the error handling, it is not included in this recipe.)

Discussion

The first step to providing user-friendly messages with your exceptions is to create a new class that inherits from `System.ApplicationException`. (The `System.ApplicationException` class extends

System.Exception but adds no new functionality. It is meant to be used to differentiate between exceptions defined by applications and those defined by the system.)

> If the custom exception classes are intended for use only in your current application, the classes should be placed in the App_Code folder of your application. If you are using Visual Studio 2005, it will offer to create the folder and place the class in it for you.
>
> If the custom exception classes are intended to be used in multiple applications, a separate Class project should be created. The assembly created by the Class project should then be placed in the bin folder of your project and a reference added to the assembly to allow using the classes in your application.

You then need to add a property to the class to support the user-friendly message. The last step in creating the new exception class is to create a constructor that will create the base exception, by calling the base class constructor with the raw message and a reference to the inner exception, and then to set the user-friendly message. Examples 8-10 (VB) and 8-11 (C#) show how we have implemented these steps.

The new exception class is put to use in the Catch block of your code by creating an instance of the new exception class, passing it the original message, a reference to the exception, and the desired user-friendly message. The reference to the original exception is passed to preserve the linked list of exceptions. In this case, your new exception will point to the original exception by using the inner property of the new exception. After the new exception class is created, it is thrown. Examples 8-12 (VB) and 8-13 (C#) illustrate a sample Catch block.

As shown in Examples 8-14 (VB) and 8-15 (C#), our sample Application_Error event handler writes detailed information to the event log and then displays the message contained in the userFriendlyMessage property of the exception. This example event code is a variation of the event code described in Recipe 8.3, modified to check if the exception being processed has a user-friendly message to use instead of the raw exception message.

This recipe's approach can be extended many ways to suit your needs. For example, the custom exception class could contain a nextPage property set to pass information on where the user should be taken after reviewing the error message.

See Also

Recipe 8.3

Example 8-10. Custom exception class with user-friendly message property (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides an exception class with support for a user-
 ''' friendly message
 ''' </summary>
 Public  Class  CH08FriendlyExceptionVB
  Inherits System.ApplicationException

  'private copy of user friendly message
  Private mUserFriendlyMessage As String = ""
  '''*************************************************************************
  ''' <summary>
  ''' Provides access to the message to be displayed to the user-friendly
  ''' message.
  ''' </summary>
  Public Property userFriendlyMessage( ) As String
   Get
    Return (mUserFriendlyMessage)
   End Get

   Set(ByVal Value As String)
    mUserFriendlyMessage = Value
   End Set
  End Property 'userFriendlyMessage

  '''*************************************************************************
  ''' <summary>
  ''' Provides a constructor supporting an error message, a reference to
  ''' the exception that threw this exeception, and a user-friendly
  ''' message for the exception
  ''' </summary>
  Public Sub New(ByVal message As String, _
       ByVal inner As Exception, _
       ByVal userFriendlyMessage As String)
   'call base class constructor. NOTE: This must be the first line in
   'this constructor
   MyBase.New(message, inner)
   mUserFriendlyMessage = userFriendlyMessage
  End Sub 'New
 End  Class  'CH08FriendlyExceptionVB
End Namespace
```

Example 8-11. Custom exception class with user-friendly message property (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides an exception class with support for a user-
 /// friendly message
 public class CH08FriendlyExceptionCS : System.ApplicationException
 {
  // private copy of user friendly message
  private String mUserFriendlyMessage = "";
  ///*********************************************************************
  /// <summary>
  /// Provides access to the message to be displayed to the user-friendly
  /// message.
  /// </summary>
  public String userFriendlyMessage
  {
   get
   {
    return(mUserFriendlyMessage);
   }
   set
   {
    mUserFriendlyMessage = value;
   }
  } // userFriendlyMessage

  ///*********************************************************************
  /// <summary>
  /// Provides a constructor supporting an error message, a reference to
  /// the exception that threw this exeception, and a user-friendly
  /// message for the exception
  /// </summary>
  public CH08FriendlyExceptionCS(String message,
          Exception inner,
          String userFriendlyMessage) :
      base(message, inner)
  {
   mUserFriendlyMessage = userFriendlyMessage;
  }
 } // CH08FriendlyExceptionCS
}
```

Example 8-12. Creation of new custom exception (.vb)

```
'''*************************************************************************
''' <summary>
''' This routine provides the event handler for the page load event. It
''' is responsible for initializing the controls on the page.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Private Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
 Dim values As Hashtable = Nothing
 Try
  'add a key/value pair to the hashtable without first creating
  'the hashtable which will cause a null exception error
  values.Add("Key", "Value")
 Catch exc As Exception
  Throw New CH08FriendlyExceptionVB(exc.Message, _
         exc, _
         "The application is currently " &_
         "experiencing technical " &_
         "difficulties … " &_
         "Please try again later")
    End Try
End Sub 'Page_Load
```

Example 8-13. Creation of new custom exception (.cs)

```
///*************************************************************************
/// <summary>
/// This routine provides the event handler for the page load event.
/// It is responsible for initializing the controls on the page.
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void Page_Load(object sender, EventArgs e)
{
 Hashtable values = null;

 try
 {
  // add a key/value pair to the hashtable without first creating
  // the hashtable which will cause a null exception error
  values.Add("Key", "Value");
 }
 catch (Exception exc)
 {
  throw new CH08FriendlyExceptionCS(exc.Message,
         exc,
         "The application is currently " +
         "experiencing technical " +
         "difficulties … " +
         "Please try again later");
    }
} // Page_Load
```

Example 8-14. Application_Error code for displaying a user-friendly message (.vb)

```
'''***************************************************************************
''' <summary>
''' This routine provides the event handler for the application error
''' event. It is responsible for processing errors at the application
''' level.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
 Const EVENT_LOG_NAME As String = "Application"

 Dim lastException As Exception
 Dim userFriendlyException As CH08FriendlyExceptionVB
 Dim Log As EventLog
 Dim message As String

 'get the last error that occurred
 lastException = Server.GetLastError( )

 'create the error message from the message in the last exception along
 'with a complete dump of all of the inner exceptions (all exception
 'data in the linked list of exceptions)
 message = lastException.Message &_
   vbCrLf &vbCrLf &_
   lastException.ToString( )

 'Insert error information into the event log
 Log = New EventLog
 Log.Source = EVENT_LOG_NAME
    Log.WriteEntry(message, _

 EventLogEntryType.Error)
 'perform other notifications, etc. here

 'check to if the exception has a user friendly message
 If (TypeOf (lastException) Is CH08FriendlyExceptionVB) Then
  'exception has a user friendly message
  userFriendlyException = CType(lastException, _
        CH08FriendlyExceptionVB)
  message = userFriendlyException.userFriendlyMessage
 Else
  'exception does not have a user friendly message to just
  'output the raw message
  message = lastException.Message
 End If

 'clear the error and redirect to the page used to display the
 'error information
 Server.ClearError( )
 Server.Transfer("CH08DisplayErrorVB.aspx"  &_
    "?PageHeader=Error Occurred" &_
    "&Message1=" &message &_
    "&Message2=" &_
    This exception used a user friendly mesage")
End Sub 'Application_Error
```

Example 8-15. Application_Error code for displaying a user-friendly message (.cs)

```
///***************************************************************************
/// <summary>
/// This routine provides the event handler for the application error
/// event. It is responsible for processing errors at the application
/// level.
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
 protected void Application_Error(Object sender, EventArgs e)
 {
  const String EVENT_LOG_NAME = "Application";

  Exception lastException = null;
  CH08FriendlyExceptionCS userFriendlyException = null;
  EventLog log = null;
  String message = null;

  // get the last error that occurred
  lastException = Server.GetLastError( );
  // create the error message from the message in the last exception along
  // with a complete dump of all of the inner exceptions (all exception
  // data in the linked list of exceptions)
  message = lastException.Message +
      "\r\r" +
      astException.ToString( );

  // Insert error information into the event log
  log = new EventLog( );
  log.Source = EVENT_LOG_NAME;
  log.WriteEntry(message,
    EventLogEntryType.Error);

  // perform other notifications, etc. here

  // check to if the exception has a user friendly message
  if (lastException.GetType( ) == typeof(CH08FriendlyExceptionCS))
  {
   // exception has a user friendly message
   userFriendlyException = (CH08FriendlyExceptionCS)(lastException);
   message = userFriendlyException.userFriendlyMessage;
  }
  else
  {
   // exception does not have a user friendly message to just
   // output the raw message
   message = lastException.Message;
  }
  // clear the error and redirect to the page used to display the
  // error information
  Server.ClearError( );
   Server.Transfer("CH08DisplayErrorCS.aspx"  +
    "?PageHeader=Error Occurred" +
    "&Message1=" + message +
    "&Message2=" +
    "This exception used a user friendly mesage");
} // Application_Error
```

PREV

Chapter 9. Security

PREV

9.0 Introduction

ASP.NET provides an infrastructure for authentication and authorization that will meet most of your needs for securing an application. Three authentication schemes are available:Forms, Windows, and Passport.

Forms

With `Forms authentication`, you use a classic custom login page to gather credentials from users and to authenticate the information supplied against a database or other data store of authorized users. You can leverage the `FormsAuthentication` APIs built into ASP.NET to issue a cookie back to the client. Recipes in this chapter show you how to use Forms authentication to restrict access to some or all pages of an application. We will show you how to restrict access to pages depending on the role assigned to the user.

Windows

Implementing `Window authentication` involves using a standard Windows dialog box to gather user credentials and validating the user against existing Windows accounts. If your application runs on an intranet, you will find that the fourth recipe in the chapter helps you implement Windows authentication in record time.

Passport

`Passport authentication` uses Microsoft's Passport service to perform the required authentication. We haven't provided any examples in this chapter, not because Passport authentication is especially difficult but because we doubt many readers are implementing it. Irrespective of our personal views, we have yet to see much interest in Passport authenticatior on a commercial level.

If none of the built-in authentication schemes provided by ASP.NET meets the needs of your application, the .NET Framework provides the ability to create your own authentication scheme. This typically involves writing a custom class that implements the`IAuthenticationModule` interface and registering it to bypass the built-in .NET authentication. Custom authentication is not covered in this book because of its individual nature. You can find more details in the MSDN Library by searching for the term "custom authentication."

This chapter provides several recipes for securing your applications using the built-in mechanisms provided by ASP.NET. These are usually adequate to meet the needs of your application.

One of the most important recommendations we can make is that you always have the security features of your application reviewed by key project stakeholders and security specialists. Bringing other perspectives to issues of security is always a good idea because it is difficult to conceive of all the ways security may be breached in your environment. Having others inspect your plans saves you

having to shoulder the entire security burden alone, which is an unwise and uncomfortable position t
be in.

With regard to the enhancements to the security-related features in ASP.NET 2.0, you will notice tha
we have made strategic use of the login controls (especially the`asp:Login` and `asp:LoginName`), which
simplify the process of writing security-related code as it relates to having users log in, log out,
provide their name, etc. We will show you how to secure your website using ASP.NET 2.0's
Membership and Role providers, which is the subject of the final recipe in the chapter. As part of this
recipe, we will show you how to use the configuration tool ASP.NET 2.0 offers for managing users an
their roles. Taken together, these security-related features make it easy to add strong authentication
mechanisms to your website with little or no code.

PREV

PREV

Recipe 9.2. Restricting Access to All Application Pages

Problem

You want to restrict access to the pages of your application to authorized users only.

Solution

Change the *web.config* settings of your application to specify Forms authentication, and create an *.aspx* l and complete the authentication check.

Modify *web.config* as follows:

1. Set the `mode` attribute of the `<authentication>` element to `Forms` .
2. Add a `<forms>` child element to the `<authentication>` element to specify key aspects of the Forms i

```
<configuration>
 <system.web>

  …

 <authentication mode="Forms">
   <forms  name=".ASPNETCookbookVBSecurity91"
    loginUrl="Login.aspx"
    protection="All"
     timeout="30"
    path="/">
  </forms>
 </authentication>
 </system.web>
</configuration>
```

3. Add `<deny>` and `<allow>` child elements to the `<authorization>` element to deny access to anonymo have been authenticated:

```
<configuration>
 <system.web>

  …
```

```
        <authorization>
         <deny users="?" /> <!-- Deny anonymous user -->
         <allow users="*" /> <!-- Allow all authenticated users -->
        </authorization>
        </system.web>
       </configuration>
```

In the *.aspx* file for the login page:

1. Add a Login control.
2. Customize the Login control as required by your application.

In the code-behind class for the login page, use the .NET language of your choice to:

1. Use the Authenticate event handler of the Login control to verify the user credentials.
2. If the user credentials are valid, create a Forms authentication cookie and add it to the cookie colle calling the `SetAuthCookie` method of the `FormsAuthentication` class.
3. (Optional) Set the Forms authentication cookie to be persisted on the client machine.
4. Redirect the user to the appropriate application start page using `Response.Redirect` .

The code we've created to illustrate this solution is shown in Examples 9-1, 9-2 , 9-3 through 9-4 . Exam make to *web.config* to restrict access to all pages. Example 9-2 shows the *.aspx* file for the login page. Ex show the code-behind class for the login page. Figure 9-1 shows the login page produced by the applicati

Figure 9-1. Example login page

Discussion

ASP.NET runs within the context of IIS and all requests must first pass through IIS, so setting up the se

always starts with setting up security in IIS. For this recipe, we do not want IIS to perform any authenti
authentication. Therefore, the website (or virtual directory) must be set up to allow anonymous access.
up anonymous access in IIS since it is easy to do and is documented in MSDN. Just search for "IIS auth

The first step to restricting access to all pages of an application is to enable ASP.NET security. This is do
the `<authentication>` element of the *web.config* file to `Forms` . (Other options are `Windows` and `Passport` .

The second step is to add a `<forms>` child element to the `<authentication>` element to specify the details
`<forms>` element has the following attributes:

`name`

Defines the name of the HTTP cookie used by ASP.NET to maintain the user authentication inform
naming the cookie. If two applications on the same server use the same cookie name, "cross auth

`loginUrl`

Defines the page to which ASP.NET will redirect users when they attempt to access pages in your
The login page should provide the fields required to authenticate the user, typically a login ID and
application requires.

`protection`

Defines the protection method used for the cookie. Possible values are`All, None, Encryption` , an
that the cookie data will be validated to ensure it was not altered in transit.`Encryption` specifies th
specifies that data validation and encryption will be used.`None` specifies no protection will be provic
default is `All` and is recommended because it offers the highest level of protection for this authen

`timeout`

Defines the amount of time in minutes before the cookie expires. The value provided here should
the session. Making the value shorter than the session timeout can result in a user being redirecte
`loginUrl` before the session times out.

`path`

Defines the path of cookies issued by the application. Be aware that most browsers treat the path
the cookie for a request that does not match the value provided for the path attribute. The result
if they were not logged in. Unless your application requires specifying the path, we recommend th

`requireSSL`

Defines if an SSL connection is required to transmit the authentication cookie. The default value is

`slidingExpiration`

Defines if sliding expiration of the authentication cookie is enabled. If set to `true` , the authenticatio
refreshed on every page request. If set to `false` , the authentication cookie will expire at the time
The default value in ASP.NET 1.0 is `true` . In ASP.NET 1.1 and 2.0, the default value is `false` .

`defaultUrl`

Defines the default URL used for redirection after authentication. The default value is *default.aspx* .
2.0.)

`Cookieless`

Defines if cookies are used to store the authentication ticket and defines the behavior of the cooki
`UseUri, AutoDetect` , and `UseDeviceProfile` . `UseCookies` specifies that cookies will always be usec
`UseUri` specifies that the authentication ticket will always be placed in the URL. For example:

http://localhost/aspnetcookbook2vb/VBSecurity91/(F(ixspTvd6lDSWeKLUrgFVdGet3iGNV6A9
rEuOlBOm4EA_HJQ128T8GJWBFZB7Hf-JwdE5oAc-Bo4d4Es1))/Home.aspx

`AutoDetect` specifies cookies will be used if the requesting device supports cookies and cookies are
supported or not enabled, the URL will be used for the authentication ticket. `UseDeviceProfile` spec
authentication ticket will be determined by the capabilities of the requesting device. This is similar
made to determine if cookies are enabled. The determination is made by the device support. The c
backward compatibility with ASP.NET 1.x. (This attribute is new for ASP.NET 2.0.)

Here is an example of the `<forms>` element and its attributes:

```
<authentication mode="Forms">
 <forms name=".ASPNETCookbookVBSecurity91"
  loginUrl="Login.aspx"
  protection="All"
  timeout="30"
  path="/" >
 </forms>
</authentication>
```

The next step is to modify *web.config* to deny access to all anonymous users and allow access to all user
This is done by adding `<deny>` and `<allow>` child elements to the `<authorization>` element:

```
<authorization>
 <deny users="?" /> <! Deny anonymous users >
 <allow users="*" /> <! Allow all authenticated users >
</authorization>
```

Your application login page should use the `asp:Login` control or provide the fields required to enter the d
This is typically a login ID and password, which you gather via text boxes, but can be whatever your app

the ability to persist the authentication cookie on the client machine. If your application supports "auto l
should provide a checkbox for the user to indicate that she wants to be remembered between sessions.
include a button to initiate the login process after the data has been entered. How we have done this fo
9-1 and in the *.aspx* file (Example 9-2).

In the code-behind for the login page, use the `Authenticate` event handler for the `asp:Login` control (or t
created your own) to verify the user credentials. In Examples 9-3 and 9-4 , for example, the database is
entered login ID and password using a `DataCommand` and a `DataReader` . After the `DataReader` is created, t
positioned before the first record in the reader. By calling the `Read` method in the `If` statement, you can
are found and read the user credentials from the database at the same time. If the user credentials are
method will return `TRue` . Otherwise, it will return `False` .

If the user credentials are valid, the Forms authentication cookie will need to be created and added to th
browser. This can be done by calling the `SetAuthCookie` method of the `FormsAuthentication` class, passin
value indicating if the value should be persisted on the client machine. To persist the authentication coo
application on subsequent sessions without logging in, set the second parameter to `true` . If the second p
authentication cookie is stored in memory on the client and is destroyed when the session expires or the

Because the `SetAuthCookie` method is static, it is unnecessary to create a `FormAuthenti`
method. The `SetAuthCookie` method is the simplest approach to creating and adding th
collection but is inflexible. For an example of a more flexible approach that allows you
authentication cookie, see Recipe 9.3.

After the application has created the authentication cookie, the user should be redirected to the appropr
redirects the user to the login page defined in your *web.config* file, it will automatically append the name
the redirected URL, as shown next. You can use this information to redirect users to the originally reque
page as illustrated in Examples 9-3 (VB) and 9-4 (C#).

```
http://localhost/ASPNetBook/Login.aspx?ReturnUrl=Home.aspx
```

You must use `Response.Redirect` to redirect the user to the next page. `Response.Redirect` returns inform
to redirect to the indicated page. This round trip to the client browser writes the authentication cookie to
the server on subsequent page requests. This cookie is what ASP.NET uses to determine if the user has
mechanisms, such as `Server.Transfer` , the authentication cookie will not be written to the browser; this
redirected back to the login page.

ASP.NET provides, in a single method call (the `RedirectFromLoginPage` method of the `Fo`
the ability to create the authentication cookie, add it to the cookie collection, and redir
original page. By default, this will redirect the user to *default.aspx* . If your site uses a
`defaultUrl` attribute of the `<forms>` element to the required filename.

No other code is required. In other words, by using the code (without adding code to each page, as is re
of the pages in your application is restricted to logged-in users.

So how does ASP.NET do this so easily? When your application is configured to use Forms authenticatio
defined by the `name` attribute of the `<forms>` element in *web.config* for every page requested from your a

exist, ASP.NET will assume the user is not logged in and will redirect the user to the page defined by the
does exist, ASP.NET will assume the user is authenticated and pass the request on to the requesting pa
exists, ASP.NET will create a user principal object with the information found in the authentication cook
principal object for short) represents the security context under which code is running. This information
accessing the `User` object in the current context. To get the username you added to the authentication c
here:

```
userName = Context.User.Identity.Name
```

```
userName = Context.User.Identity.Name;
```

Applications that provide the ability to log inparticularly those that provide the ability to persist the infor
eliminate the need to log in on subsequent visitsshould provide the ability to log out. In this context, log
cookie on the client machine, which requires the user to log in again to gain access to your application.
line of code shown here:

```
FormsAuthentication.SignOut()
```

```
FormsAuthentication.SignOut();
```

ASP.NET can provide security only for files mapped to the ASP.NET ISAPI DLL. By defa
extensions *.asax, .ascx, .ashx, .asmx, .aspx, .axd, .vsdisco, .rem, .soap, .config, .cs,*
.webinfo, .licx , and *.resx* . Any other file typessuch as *.gif, .jpg, .txt* , and *.js* are not p
security. If access to these file types must be restricted, they will have to be added to
processed by the ASP.NET ISAPI DLL. This can be done in the application configuratio
properties of your application. Requiring these file types to be processed by ASP.NET v
your application, because of the extra processing required for the images, text files, Ja

See Also

Recipe 9.3; MSDN documentation for IIS setup (search for "IIS authentication")

Example 9-1. Changes to web.config to restrict access to all pages

```
<?xml version="1.0"?>
<configuration>
 <system.web>

  …

  <authentication mode="Forms">
   >forms name=".ASPNETCookbookVBSecurity91"

   loginUrl="Login.aspx"
   protection="All"
   timeout="30"
   path="/" >
  </forms>
 </authentication>

 <authorization>
  <deny users="?" /> <!-- Deny anonymous user -->
  <allow users="*" /> <!-- Allow all authenticated users -->
 </authorization>

  …
  </system.web>
</configuration>
```

Example 9-2. Login page (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="Login.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.Login"
 Title="Login" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Block Access To All Pages (VB)
 </div>
 <div align="center">
  <asp:Login ID="Login1" runat="server"
    TitleText=""
    UserNameLabelText="Login ID: "
    PasswordLabelText="Password: "
    DisplayRememberMe="true"
    RememberMeText="Remember Me"
    LabelStyle-CssClass="labelText"
    CheckBoxStyle-CssClass="labelText"
    TextBoxStyle-CssClass="labelText"
    OnAuthenticate="Login1_Authenticate" >
   <CheckBoxStyle CssClass="labelText" />
   <TextBoxStyle CssClass="labelText" />
   <LabelStyle CssClass="labelText" />
  </asp:Login>
  <br />
  <input type="button" value="Attempt Access without Login"
    onclick="document.location='Home.aspx'" />
 </div>
</asp:Content>
```

Example 9-3. Login page code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' Login.aspx
 ''' </summary>
 Partial Class Login
  Inherits System.Web.UI.Page

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the login button click
  ''' event. It is responsible for authenticating the user and redirecting
  ''' to the next page if the user is authenticated.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Login1_Authenticate(ByVal sender As Object, _
         ByVal e As AuthenticateEventArgs)
   'name of querystring parameter containing return URL
   Const QS_RETURN_URL As String = "ReturnURL"
   Dim dbConn As OleDbConnection = Nothing
   Dim dCmd As OleDbCommand = Nothing
   Dim dr As OleDbDataReader = Nothing
   Dim strConnection As String
   Dim strSQL As String
   Dim nextPage As String

   Try
    'get the connection string from web.config and open a connection
    'to the database
    strConnection = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDb.OleDbConnection(strConnection)
    dbConn.Open( )

    'check to see if the user exists in the database
    strSQL = "SELECT (FirstName + ' ' + LastName) AS UserName " & _
     "FROM AppUser " & _
     "WHERE LoginID=? AND " & _
     "Password=?"
    dCmd = New OleDbCommand(strSQL, dbConn)
    dCmd.Parameters.Add(New OleDbParameter("LoginID", _
           Login1.UserName))
    dCmd.Parameters.Add(New OleDbParameter("Password", _
           Login1.Password))
    dr = dCmd.ExecuteReader( )

    If (dr.Read( )) Then
    'user credentials were found in the database so notify the system
    'that the user is authenticated
    FormsAuthentication.SetAuthCookie(CStr(dr.Item("UserName")), _
          Login1.RememberMeSet)

    'get the next page for the user
    If (Not IsNothing(Request.QueryString(QS_RETURN_URL))) Then
    'user attempted to access a page without logging in so redirect
    'them to their originally requested page
    nextPage = Request.QueryString(QS_RETURN_URL)
    Else
    'user came straight to the login page so just send them to the
    'home page
    nextPage = "Home.aspx"
    End If

    'Redirect user to the next page
    'NOTE: This must be a Response.Redirect to write the cookie to the
    ' user's browser. Do NOT change to Server.Transfer which
    ' does not cause around trip to the client browser and thus
    ' will not write the authentication cookie to the client
    '   browser.
    Response.Redirect(nextPage, True)
    Else
    'user credentials do not exist in the database so output error
    'message indicating the problem
    Login1.FailureText = "Login ID or password is incorrect. " & _
       "Please check your credentials and try again."
    End If
```

```
    Finally
     'cleanup
     If (Not IsNothing(dr)) Then
     dr.Close( )
     End If

     If (Not IsNothing(dbConn)) Then
     dbConn.Close( )
     End If
    End Try
   End Sub 'Login1_Authenticate
 End Class 'Login
End Namespace
```

Example 9-4. Login page code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.Security;
using System.Web.UI.WebControls;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 /// Login.aspx
 /// </summary>
 public partial class Login : System.Web.UI.Page
 {
  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the login button click
  /// event. It is responsible for authenticating the user and redirecting
  /// to the next page if the user is authenticated.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Login1_Authenticate(Object sender,
          AuthenticateEventArgs e)
  {
   // name of querystring parameter containing return URL
   const String QS_RETURN_URL = "ReturnURL";

   OleDbConnection dbConn = null;
```

```
OleDbCommand dCmd = null;
OleDbDataReader dr = null;
String strConnection = null;
String strSQL = null;
String nextPage = null;

try
{

 // get the connection string from web.config and open a connection
 // to the database
 strConnection = ConfigurationManager.
 ConnectionStrings["dbConnectionString"].ConnectionString;
 dbConn = new OleDbConnection(strConnection);
 dbConn.Open( );

 // check to see if the user exists in the database
 strSQL = "SELECT (FirstName + ' ' + LastName) AS UserName " +
 "FROM AppUser " +
 "WHERE LoginID=? AND " +
 "Password=?";
 dCmd = new OleDbCommand(strSQL, dbConn);
 dCmd.Parameters.Add(new OleDbParameter("LoginID",
         Login1.UserName));
 dCmd.Parameters.Add(new OleDbParameter("Password",
         Login1.Password));
 dr = dCmd.ExecuteReader( );

if (dr.Read( ))
{
// user credentials were found in the database so notify the system
// that the user is authenticated
FormsAuthentication.SetAuthCookie((String)(dr["UserName"]),
        Login1.RememberMeSet);

// get the next page for the user
if (Request.QueryString[QS_RETURN_URL] != null)
{
 // user attempted to access a page without logging in so redirect
 // them to their originally requested page
 nextPage = Request.QueryString[QS_RETURN_URL];
}
else
{
 // user came straight to the login page so just send them to the
 // home page
 nextPage = "Home.aspx";
}

// Redirect user to the next page
// NOTE: This must be a Response.Redirect to write the cookie to
//  the user's browser. Do NOT change to Server.Transfer
```

```
    //  which does not cause around trip to the client browser
    //  and thus will not write the authentication cookie to the
    //  client browser.
    Response.Redirect(nextPage, true);
   }
   else
   {
    // user credentials do not exist in the database so output error
    // message indicating the problem
    Login1.FailureText = "Login ID or password is incorrect. " +
      "Please check your credentials and try again.";
   }
  } // try

  finally
  {
   // cleanup
   if (dr != null)
   {
    dr.Close( );
   }

   if (dbConn != null)
   {
    dbConn.Close( );
   }
  } // finally

  }  // Login1_Authenticate
 } // Login
 }
```

Recipe 9.3. Restricting Access to Selected Application Pages

Problem

You want to restrict access to many of the pages in your application (i.e., you want to make some pages accessible to the public).

Solution

Implement the solution described in Recipe 9.1 and modify the contents of the *web.config* file to list the pages allowing public access and requiring authentication.

Modify *web.config* as follows:

1. Change the `<deny>` child element of the `<authorization>` element to `<deny users="*"/>` and delete the `<allow>` child element to deny access to all users.
2. Add a `<location>` element to the configuration level for each application page to specify if it is available to the public or only to authenticated users.

Example 9-5 shows how we have implemented this solution with some sample *web.config* entries. We begin by adding settings that deny access to all users. We then add settings that allow public access to *PublicPage.aspx* but restrict access to *Home.aspx* only to authenticated users.

Discussion

The approach we advocate for this recipe is the same as for Recipe 9.1, except for certain aspects of the *web.config* file configuration.

The `<authentication>` element and its `<forms>` child are the same as in Recipe 9.1.

We have modified the `<authorization>` element that we used in Recipe 9.1 to deny access to all users. By denying authorization to all users at the application level, elements can be added to authorize access to particular pages.

Access to the individual pages in the application is controlled by providing a `<location>` element for each page. For pages with public access, the `<location>` element should be set up as follows:

```
<location path="PublicPage.aspx">
 <system.web>
  <authorization>
   <allow users="*"/>
  </authorization>
 </system.web>
</location>
```

The path attribute of the <location> element specifies the page to which this <location> element applies. The <authorization> element defines the access to the page defined in thepath attribute. For public access, the <authorization> should be set to allow all users (users="*").

To limit access to individual pages, a<location> element is provided for each of the restricted pages, as follows:

```
<location path="Home.aspx">
 <system.web>
  <authorization>
   <deny users="?" /> <!-- Deny anonymous (unauthenticated) users -->
   <allow users="*"/> <!-- Allow all authenticated users -->
  </authorization>
 </system.web>
</location>
```

The primary difference is the inclusion of two<authorization> child elements. The <deny> element denies access to anonymous (unauthenticated) users. The<allow> element allows access to all authenticated users. We've used two special identities in the code: ? and *. Here ? refers to the anonymous identity, and * refers to all identities.

At first glance, this code sequence would appear to allow access to all users. ASP.NET processes authentication and authorization in a hierarchical fashion, starting at*machine.config*, followed by the *web.config* for the application and any *web.config* files located in folders in the path to the requested page. When ASP.NET finds the first access rule that applies to the current user, the rule will be applied. The rule evaluation continues, and if any other rules are found that reduce the access, they will be applied; otherwise, they will be skipped. In this case, the<deny users="?" /> rule is processed first. The <allow users="*"/> rule is processed second, and because it does not reduce the access, it is skipped. Therefore, if the user is not authenticated, he is denied access. As you can see by this example, the precise ordering of the <allow> and <deny> elements is important. They must be in most restrictive order, from top to bottom.

One of the disadvantages of using a <location> element for each page you wish to make accessible to authenticated users is the maintenance of all the page names in the<location> elements. An application with 50 pages is easy to process; however, an application with a large number of pages can be a big task. ASP.NET provides another mechanism, described later, to make the process a littl
easier.

Though this approach may have some maintenance issues, it does have the advantage of providing more control over the individual pages served by your application. For instance, without modifying the *web.config* file, no one will be able to add a page to your application that will be viewable through the web server. If someone does succeed in adding a page, attempts to access it will result in a redirection to the login page.

If the idea of maintaining a `<location>` element for each page in your application is unappealing, you can structure your application to place public pages and private pages in separate folders and then provide one `<location>` element for the public folder and a second one for the private pages:

```
<!--
****************************************************************************
 The following location element provides public access to all pages
 in the PublicPages folder.
****************************************************************************
-->
<location path="PublicPages">
 <system.web>
  <authorization>
   <allow users="*"/>
  </authorization>
 </system.web>
</location>

<!--
****************************************************************************
 The following location element restricts access to all pages
 in the PrivatePages folder.
****************************************************************************
-->
<location path="PrivatePages">
 <system.web>
  <authorization>
   <deny users="?" /> <!-- Deny anonymous (unauthenticated) users -->
   <allow users="*"/> <!-- Allow all authenticated users -->
  </authorization>
 </system.web>
</location>
```

Using `<location>` elements to define separate folders for public and private pages is analogous to using them to set authorization requirements for individual pages, except the path attribute is set to a relative path to an applicable folder rather than to a specific*.aspx* page.

Nothing comes free. The folder approach to controlling security makes the maintenance of the *web.config* file simpler; however, it has two drawbacks. First, any page placed in a designated folder becomes part of your application whether you want it to or not. Second, sharing images and user controls is more difficult. This is the result of having to provide a relative path from the requested page to the image or user control. It becomes more difficult if a user control contains images and is used in the root folder as well as in a subfolder.

See Also

Recipe 9.1; MSDN documentation for more information on*web.config* format (search for "Format of ASP.NET Configuration Files")

Example 9-5. web.config entries to restrict access to some pages

```
<?xml version="1.0"?>
<configuration>
 <system.web>

  …
  <authentication mode="Forms">
    <forms name=".ASPNETCookbookVBSecurity921"
     loginUrl="Login.aspx"
     protection="All"
     timeout="30"
     path="/">
   </forms>
  </authentication>

  <authorization>
   <deny users="*" /> <!-- Deny all users -->
  </authorization>

  …
 </system.web>

 <!--
 ****************************************************************************
  The following section provides public access to pages that do not
  require authentication. An entry must be included for each page
  or folder that does not require authentication.
 ****************************************************************************
 -->
 <location path="PublicPage.aspx">
  <system.web>
   <authorization>
```

```
    <allow users="*"/>
   </authorization>
  </system.web>
 </location>

 <!--
 ****************************************************************************
  The following section defines the pages that require authentication
  for access. An entry must be included for each page or folder that
  requires authentication.
 ****************************************************************************

 -->
 <location path="Home.aspx">
  <system.web>
   <authorization>
    <deny users="?" /><!-- Deny anonymous users -->
    <allow users="*"/><!-- Allow all authenticated users -->
   </authorization>
  </system.web>
 </location>

</configuration>
```

Recipe 9.4. Restricting Access to Application Pages by Role

Problem

You want to assign or make use of predefined roles for the users of your application, and you want to control access to pages as a function of these roles.

Solution

The solution involves the following four steps:

1. Implement the solution described in Recipe 9.2, adding the required roles to *web.config* for each of the pages.
2. In the code-behind class for the ASP.NET login page, add the user's role information to the authentication cookie when the user logs in.
3. Add code to the `Application_AuthenticateRequest` method in the *global.asax* file to recover the user role information and build a user principal object.
4. Set the user principal object to the `Context.User` property to provide ASP.NET the data it needs to perform page-by-page authentication.

The code we've written to illustrate this solution appears in Examples 9-6, 9-7, 9-8, 9-9 through 9-10. The `<authentication>` and `<authorization>` elements of *web.config* are shown in Example 9-6. The login page code-behind where the authentication cookie is created is shown in Examples 9-7 (VB) and 9-8 (C#). (See Recipe 9.1 for the *.aspx* file for a typical login page.) The `Application_AuthenticateRequest` method in the code-behind for *global.asax* is shown in Examples 9-9 (VB) and 9-10 (C#).

Discussion

The approach we favor for this recipe builds on Recipe 9.2 but takes a tack of its own based on the addition and use of user roles. The `<authentication>` and `<authorization>` elements of the *web.config* file are identical to those used in Recipe 9.2. And, as in Recipe 9.2, `<location>` elements are used to define the access requirements for each page. The `<location>` elements for the public access pages are also identical.

In this recipe, however, each `<location>` elements for the restricted pages contains a list of roles required for access to the page it controls. The following code shows an example. For*Home.aspx*, the User and `Admin` roles are allowed access. For *AdminPage.aspx*, only the `Admin` role is allowed access:

```
  <location path="Home.aspx">
 <system.web>
   <authorization>
  <allow roles="User,
    Admin"/>
  <deny users="*"/>
   </authorization>
  </system.web>
  </location>

  <location path="AdminPage.aspx">
 <system.web>
  <authorization>
   <allow roles="Admin"/>
   <deny users="*"/>
  </authorization>
 </system.web>
  </location>
```

Include the `<deny users="*" />` element for all pages after the list of roles allowed to access the page. This informs ASP.NET that if users are not assigned one of the previously listed roles, they should be denied access to the page.

You might be tempted to use folders to contain pages with similar access rights, as described in Recipe 9.2. However, this approach is more complicated when roles are used, because a user can be assigned multiple roles and any given page may be accessible by multiple roles. You might initially be able to segment your application to use the folder approach, but making changes later will be difficult. If you are using roles to control access to pages in your application, you should include a `<location>` element for each page in your *web.config* file.

The operations required after the user credentials have been verified are different than the previous recipes in this chapter. First, a `FormsAuthenticationTicket` is created. This authentication ticket will be used as the authentication cookie. By manually creating the ticket, you can add the user role information to the authentication cookie.

To create the ticket manually, you must instantiate a `System.Web.Security.FormsAuthenticationTicket` object, as shown in a general form here and in a more application-specific form in Examples 9-7 (VB) and 9-8 (C#):

```
 ticket = New FormsAuthenticationTicket(version, _
    name
    issueDate
    expiration
    isPersistant
    userData
```

The ticket takes six parameters:

`version`

> The first parameter is a version number. In our code-behind examples, we've used the number 1, but you can use any value. In a production application that allows persistent cookies to be stored on the client, the version number can be used to track and handle changes to the data in the cookie.

`name`

> The second parameter is the user's name. This can be any string you want to use to identify the user.

`issueDate`

> The third parameter is the issue date/time of the authentication ticket. This would normally be set to the current date/time.

`expiration`

> The fourth parameter is the expiration date/time of the authentication ticket. The difference between this value and the issue date/time needs to be greater than or equal to the session timeout value. In Examples 9-7 (VB) and 9-8 (C#), it is set to 30 minutes in the future.

`isPersistent`

> The fifth parameter is a flag indicating whether the cookie should be persisted on the client (if set to `true`) or should be memory-based (if set to `False`).

`userData`

> The sixth parameter can be any string value you want to store with the cookie. It is used in ou example to store a comma-delimited list of userroles. This list of roles is used to authorize access to the pages in your application.

> Depending on how the role information is stored for a *user* of your application, you may need to build a comma-delimited string containing theroles and use this string as the sixth parameter of the authentication ticket.

After the application creates it, an authentication ticket needs to be converted to an encrypted string Only strings can be stored in cookies, and encryption prevents the possibility of tampering with the data stored there. To encrypt the string, we call the`Encrypt` method of the `FormsAuthentication` class, passing it the ticket that we created. The method returns a string containing the encrypted ticket:

```
encryptedStr = FormsAuthentication.Encrypt(ticket)
```

Now you need to create a cookie from the encrypted string, using the name of the Forms authentication cookie defined in the *web.config* file. Naming the cookie anything else will keep the authentication from working since ASP.NET will look for the cookie by the name defined in*web.config* and will not find it.

```
cookie = New HttpCookie(FormsAuthentication.FormsCookieName,
   encryptedStr)
```

Next, you need to set the expiration date and time for the cookie. If the cookie is to be persisted, the expiration should be set in the future. In Examples 9-7 and 9-8, we've set the expiration date 10 years in the future. If the cookie is not to be persisted, the expiration date and time should not be se or the cookie will be made persistent.

The last step before redirecting the user to the appropriate next page (the same as in previous recipes in this chapter) is to add the cookie you created to the cookie collection so it will be sent to the client browser on the redirect. This is done by calling the`Add` method of the `Response` object's `Cookies` collection and passing it the cookie to be added.

Now that the user is logged in and the role information has been added to the authentication cookie, you need to add the `Application_AuthenticateRequest` method to *global.asax* (see Examples 9-9 and 9-10). This method is executed for every requested resource that ASP.NET manages and allows the authentication/ authorization process to be customized.

In the `Application_AuthenticateRequest` method, the first thing you need to do is to check if the user is currently authenticated by calling the`IsAuthenticated` method of the `Request` object. If not, no action needs to be taken.

If the user is authenticated, you need to check if the authentication type is set to`Forms`. If it is not, an exception should be thrown since a significant mismatch exists between the code and the authentication cookie.

```
If (Context.Request.IsAuthenticated) Then
  If (Context.User.Identity.AuthenticationType = "Forms") Then

 …
```

```
    Else
   'application is improperly configured so throw an exception
   Throw New ApplicationException("Application Must Be Configured For
   Forms Authentication")
   End If 'If (Context.User.Identity.AuthenticationType = "Forms")
  End If 'If (Context.Request.IsAuthenticated)
```

```
  if (Context.Request.IsAuthenticated)
  {
  if (Context.User.Identity.AuthenticationType == "Forms")
  {
   …
  }
  else
  {
   // application is improperly configured so throw an exception
   throw new ApplicationException("Application Must Be Configured For
     Forms Authentication");
    } // if (Context.User.Identity.AuthenticationType = "Forms")
  } // if (Context.Request.IsAuthenticated)
```

Next, you need to get the user roles that you added to the Forms authentication cookie. This is done by getting the identity from the `Context.User` object. You must cast the identity to a `FormsIdentity` object to get access to the authentication ticket data where the user roles are stored. This casting is the primary reason why the verification was made on the authentication type. If it was not `Forms`, this casting would throw a generic exception that would be much harder to troubleshoot than an exception that explicitly states what is wrong. The list of roles retrieved from the `UserData` property of the authentication ticket is the comma-delimited list of the user roles you added to the authentication ticket during login.

```
  identity = CType(Context.User.Identity, FormsIdentity)
  roles = identity.Ticket.UserData
```

```
  identity = (FormsIdentity)(Context.User.Identity);
  roles = identity.Ticket.UserData;
```

With the user identity and the roles in hand, you have the information you need to create a `GenericPrincipal` object and assign it to the `User` property of the `Context` object. This `GenericPrincipal` object is what ASP.NET uses, along with the information in the *web.config* file, to perform the authorization for the requested page.

The `GenericPrincipal` constructor requires the user's identity and an array of strings for the roles assigned to the user. The user's identity is the `FormsIdentity` object you retrieved previously. The comma-delimited list of roles you retrieved previously can easily be converted to an array of strings by using the `Split` method of the `String` object, passing a comma as the delimiter used for performing the split.

VB

```
Context.User = new GenericPrincipal(identity,
      roles.Split(','));
```

C#

```
Context.User = New GenericPrincipal(identity, _
      roles.Split(","))
```

As with previous recipes in this chapter, you do not need to add any code to the individual pages in your application to handle the authentication and authorization. ASP.NET will do that work for you and let you concentrate on the requirements of the individual pages. If the authorization requirements change for your application and different roles are required to access pages, the only changes required will be to the *web.config* `<location>` elements.

Though you do not need to perform any of the authentication and authorization tasks, you may want access to the information to customize your pages with the user information or to change what is displayed on pages as a function of the user's roles. The username can be obtained as shown here:

```
userName = context.User.Identity.Name
```

```
userName = context.User.Identity.Name;
```

To check the user's role(s), use the following code:

```
If (Context.User.IsInRole("User")) Then

'perform functions for role

End If
```

```
if (Context.User.IsInRole("User"))
{
 // perform functions for role
}
```

Using the `GenericPrincipal` object, no mechanism is provided to get the list of roles assigned to the user. You can only check to see if a user can perform a specific role. If your application requires that you have access to the list of roles, you can store the role information in a`Session` variable or you can create a custom user principal inheriting from the`GenericPrincipal` class and adding the functionality needed by your application.

See Also

Recipes 9.1 and 9.2

Example 9-6. web.config for restricting access by user role

```
<?xml  version="1.0"?>
<!--
<configuration>
 <system.web>

 …

 <authentication mode="Forms">
  <forms name=".ASPNETCookbookVBSecurity93"
    loginUrl="Login.aspx"
    protection="All"
    timeout="30"
    path="/">
  </forms>
 </authentication>

 <authorization>
   <!-- Deny all users -->
   <deny users="*" />
 </authorization>

 </system.web>
 <!--
 ****************************************************************************
  The following section provides public access to pages that do not
  require authentication. An entry must be included for each page
  or folder that does not require authentication.
 ****************************************************************************
 -->
 <location path="PublicPage.aspx">
  <system.web>
   <authorization>
    <allow users="*"/>
   </authorization>
  </system.web>
 </location>

 <!--
 ****************************************************************************
  The following section defines the pages that require authentication
  for access. An entry must be included for each page or folder that
  requires authentication with a list of the roles required for access
  to the page.

  Valid Roles are as follows.
  NOTE: The roles must be entered exactly as listed.

    User
    Admin
 ****************************************************************************
 -->
 <location path="Home.aspx">
  <system.web>
   <authorization>
    <allow roles="User,
     Admin"/>
    <deny users="*"/>
   </authorization>
  </system.web>
 </location>

 <location path="AdminPage.aspx">
  <system.web>
   <authorization>
    <allow roles="Admin"/>
    <deny users="*"/>
   </authorization>

  </system.web>
 </location>

</configuration>
```

Example 9-7. Login page code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb
```

```
Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' Login.aspx
 ''' </summary>
 Partial Class Login
  Inherits System.Web.UI.Page

   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the login button click
  ''' event. It is responsible for authenticating the user and redirecting
  ''' to the next page if the user is authenticated.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub Login1_Authenticate(ByVal sender As Object, _
         ByVal e As AuthenticateEventArgs)
   'name of querystring parameter containing return URL
   Const QS_RETURN_URL As String = "ReturnURL"

   Dim dbConn As OleDbConnection = Nothing
   Dim dCmd As OleDbCommand = Nothing
   Dim dr As OleDbDataReader = Nothing
   Dim strConnection As String
   Dim strSQL As String
   Dim nextPage As String
   Dim ticket As FormsAuthenticationTicket
   Dim cookie As HttpCookie
   Dim encryptedStr As String

   Try
    'get the connection string from web.config and open a connection
    'to the database
    strConnection = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString

   dbConn = New OleDb.OleDbConnection(strConnection)
   dbConn.Open( )

   'check to see if the user exists in the database
   strSQL = "SELECT (FirstName + ' ' + LastName) AS UserName, " & _
     "Role " & _
     "FROM AppUser " & _
     "WHERE LoginID=? AND " & _
     "Password=?"
   dCmd = New OleDbCommand(strSQL, dbConn)
   dCmd.Parameters.Add(New OleDbParameter("LoginID", _
          Login1.UserName))
   dCmd.Parameters.Add(New OleDbParameter("Password", _
          Login1.Password))
   dr = dCmd.ExecuteReader( )

   If (dr.Read( )) Then
   'user credentials were found in the database so notify the system
   'that the user is authenticated

   'create an authentication ticket for the user with an expiration
   'time of 30 minutes and placing the user's role in the userData
   'property
   ticket = New FormsAuthenticationTicket(1, _
       CStr(dr.Item("UserName")), _
       DateTime.Now( ), _
       DateTime.Now( ).AddMinutes(30), _
       Login1.RememberMeSet, _
       CStr(dr.Item("Role")))
   encryptedStr = FormsAuthentication.Encrypt(ticket)

   'add the encrypted authentication ticket in the cookies collection
   'and if the cookie is to be persisted, set the expiration for
   '10 years from now. Otherwise do not set the expiration or the
   'cookie will be created as a persistent cookie.
   cookie = New HttpCookie(FormsAuthentication.FormsCookieName, _
      encryptedStr)
   If (Login1.RememberMeSet) Then
    cookie.Expires = ticket.IssueDate.AddYears(10)
   End If

   Response.Cookies.Add(cookie)

   'get the next page for the user
   If (Not IsNothing(Request.QueryString(QS_RETURN_URL))) Then
    'user attempted to access a page without logging in so redirect
    'them to their originally requested page
    nextPage = Request.QueryString(QS_RETURN_URL)
   Else
    'user came straight to the login page so just send them to the
    'home page
    nextPage = "Home.aspx"
   End If

   'Redirect user to the next page
   'NOTE: This must be a Response.Redirect to write the cookie to the
   ' user's browser. Do NOT change to Server.Transfer which
   ' does not cause around trip to the client browser and thus
   ' will not write the authentication cookie to the client
   ' browser.
   Response.Redirect(nextPage, True)
  Else
   'user credentials do not exist in the database - in a production
   'application this should output an error message telling the user
   'that the login ID or password was incorrect
  End If
```

```
  Finally
   'cleanup
   If (Not IsNothing(dr)) Then
    dr.Close( )
   End If

   If (Not IsNothing(dbConn)) Then
    dbConn.Close( )
   End If
    End Try
  End Sub 'Login1_Authenticate
   End Class 'Login
End Namespace
```

Example 9-8. Login page code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.Security;
using System.Web.UI.WebControls;
using System.Web;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
  /// Login.aspx
  /// </summary>
  public partial class Login : System.Web.UI.Page
  {

   ///*************************************************************************
   /// <summary>
   /// This routine provides the event handler for the login button click
   /// event. It is responsible for authenticating the user and redirecting
   /// to the next page if the user is authenticated.
   /// </summary>
   ///

   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void Login1_Authenticate(Object sender,
         AuthenticateEventArgs e)
   {
   // name of querystring parameter containing return URL
```

```
const String QS_RETURN_URL = "ReturnURL";

OleDbConnection dbConn = null;
OleDbCommand dCmd = null;
OleDbDataReader dr = null;
String strConnection = null;
String strSQL = null;
String nextPage = null;
FormsAuthenticationTicket ticket = null;
HttpCookie cookie = null;
String encryptedStr = null;

try
{
 // get the connection string from web.config and open a connection
 // to the database
 strConnection = ConfigurationManager.
  ConnectionStrings["dbConnectionString"].ConnectionString;
 dbConn = new OleDbConnection(strConnection);
 dbConn.Open( );

 // check to see if the user exists in the database
 strSQL = "SELECT (FirstName + ' ' + LastName) AS UserName, " +
   "Role " +
   "FROM AppUser " +
   "WHERE LoginID=? AND " +
   "Password=?";
 dCmd = new OleDbCommand(strSQL, dbConn);
 dCmd.Parameters.Add(new OleDbParameter("LoginID",
        Login1.UserName));
 dCmd.Parameters.Add(new OleDbParameter("Password",
        Login1.Password));
 dr = dCmd.ExecuteReader( );

 if (dr.Read( ))
 {
 // user credentials were found in the database so notify the system
 // that the user is authenticated
 // create an authentication ticket for the user with an expiration
 // time of 30 minutes and placing the user's role in the userData
 // property
 ticket = new FormsAuthenticationTicket(1,
     (String)(dr["UserName"]),
     DateTime.Now,
     DateTime.Now.AddMinutes(30),

     Login1.RememberMeSet,
     (String)(dr["Role"]));
 encryptedStr = FormsAuthentication.Encrypt(ticket);

 // add the encrypted authentication ticket in the cookies collection
 // and if the cookie is to be persisted, set the expiration for
```

```
    // 10 years from now. Otherwise do not set the expiration or the
    // cookie will be created as a persistent cookie.
    cookie = new HttpCookie(FormsAuthentication.FormsCookieName,
       encryptedStr);
    if (Login1.RememberMeSet)
    {
     cookie.Expires = ticket.IssueDate.AddYears(10);
    }

    Response.Cookies.Add(cookie);

    // get the next page for the user
    if (Request.QueryString[QS_RETURN_URL] != null)
    {
     // user attempted to access a page without logging in so redirect
     // them to their originally requested page
     nextPage = Request.QueryString[QS_RETURN_URL];
    }
    else
    {
     // user came straight to the login page so just send them to the
     // home page
     nextPage = "Home.aspx";
    }

    // Redirect user to the next page
    // NOTE: This must be a Response.Redirect to write the cookie to
    // the user's browser. Do NOT change to Server.Transfer
    // which does not cause around trip to the client browser
    // and thus will not write the authentication cookie to the
    // client browser.
    Response.Redirect(nextPage, true);
   }
   else
   {
    // user credentials do not exist in the database so output error
    // message indicating the problem
    Login1.FailureText = "Login ID or password is incorrect. " +
      "Please check your credentials and try again.";
   }
  } // try

  finally
  {
   // cleanup
   if (dr != null)

   {
    dr.Close( );
   }

   if (dbConn != null)
   {
    dbConn.Close( );
   }
    } // finally
  } // Login1_Authenticate
   } // Login
 }
```

Example 9-9. Application_AuthenticateRequest method in global.asax (VB)

```
'''***************************************************************************
''' <summary>
''' This routine provides the event handler for the application authenticate
''' request event. It is responsible for initializing application variables.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub Application_AuthenticateRequest(ByVal sender As Object, _
          ByVal e As EventArgs)
 Dim roles As String
 Dim identity As FormsIdentity

 If (Context.Request.IsAuthenticated) Then
  If (Context.User.Identity.AuthenticationType = "Forms") Then
     'get the comma delimited list of roles from the user data
     'in the authentication ticket
     identity = CType(Context.User.Identity, FormsIdentity)
     roles = identity.Ticket.UserData

     'create a new user principal object with the current user identity
     'and the roles assigned to the user
     Context.User = New GenericPrincipal(identity, _
         roles.Split(","))
  Else
    'application is improperly configured so throw an exception
    Throw New ApplicationException("Application Must Be Configured For Forms
Authentication")
  End If 'If (Context.User.Identity.AuthenticationType = "Forms")
 End If 'If (Context.Request.IsAuthenticated)
End Sub 'Application_AuthenticateRequest
```

Example 9-10. Application_AuthenticateRequest method in global.asax

(C#)

```
///***********************************************************************
/// <summary>
/// This routine provides the event handler for the application authenticate
/// request event. It is responsible for initializing application variables.

/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void Application_AuthenticateRequest(Object sender, EventArgs e)
{
 String roles = null;
 FormsIdentity identity = null;

 if (Context.Request.IsAuthenticated)
 {
   if (Context.User.Identity.AuthenticationType == "Forms")
   {

  // get the comma delimited list of roles from the user data
  // in the authentication ticket
  identity = (FormsIdentity)(Context.User.Identity);
  roles = identity.Ticket.UserData;

  // create a new user principal object with the current user identity
  // and the roles assigned to the user
  Context.User = new GenericPrincipal(identity,
        roles.Split(','));
  }
  else
  {

    // application is improperly configured so throw an exception
    throw new ApplicationException("Application Must Be Configured For Forms
Authentication");
  } // if (Context.User.Identity.AuthenticationType = "Forms")
 } // if (Context.Request.IsAuthenticated)
} // Application_AuthenticateRequest
```

Recipe 9.5. Using Windows Authentication

Problem

You want to use existing Windows network accounts for authenticating users of your application.

Solution

Configure IIS to block anonymous access and to require Windows integrated authentication.

Make the following four changes to *web.config*:

1. Specify Windows authentication:

```
<authentication mode="Windows" />
```

2. Set the <identity> element to `impersonate`:

```
<identity impersonate="true" userName="" password="" />
```

3. Configure the `<authorization>` element to deny access to all users:

```
<authorization>
 <deny users="*" /> <!-- Deny all users -->
</authorization>
```

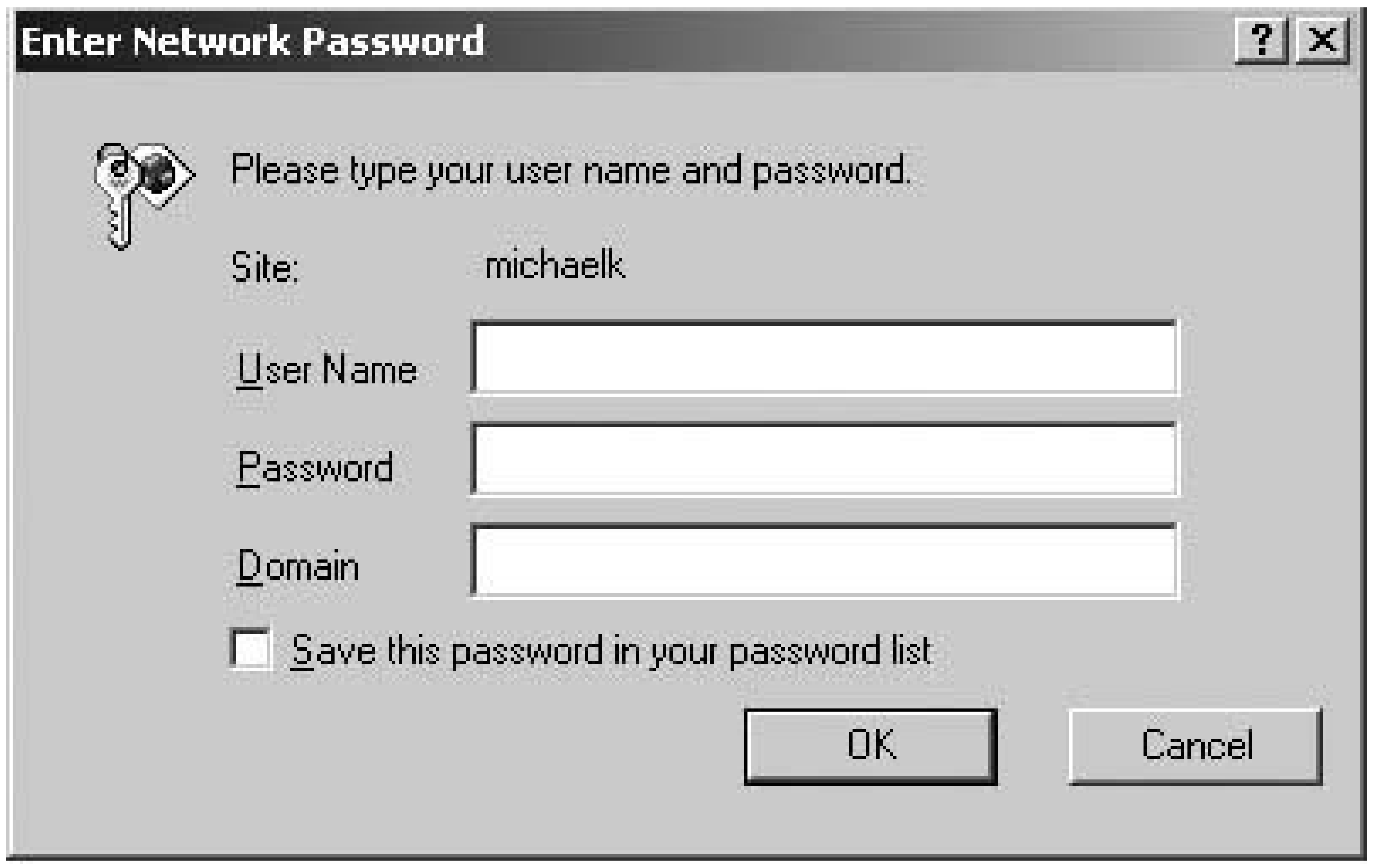
4. Add a `<location>` element for each page to which you want to control access with an `<allow>` child element and attribute (to allow access to the page by certain roles) followed by a `<deny>` child element and attribute (to deny access to all users not listed in the previous roles):

```
<location path="DisplayUserInformation.aspx">
  <system.web>
```

```
         <authorization>
        <allow roles="BuiltIn\Users,
           BuiltIn\Administrators"/>
        <deny users="*"/>
      </authorization>
       </system.web>
     </location>
```

The code we've implemented to illustrate this solution appears in Examples 9-11, 9-12, 9-13 through 9-14. Example 9-11 shows the Windows authentication and role settings in *web.config* for the sample ASP.NET page. Example 9-12 shows the Windows authentication sample *.aspx* file. The code-behind class for the page appears in Examples 9-13 (VB) and 9-14 (C#). Figure 9-2 shows the Windows authentication dialog box, and Figure 9-3 shows a sample page produced by the application.

Figure 9-2. Windows authentication dialog box


Enter Network Password

Please type your user name and password.

Site: michaelk

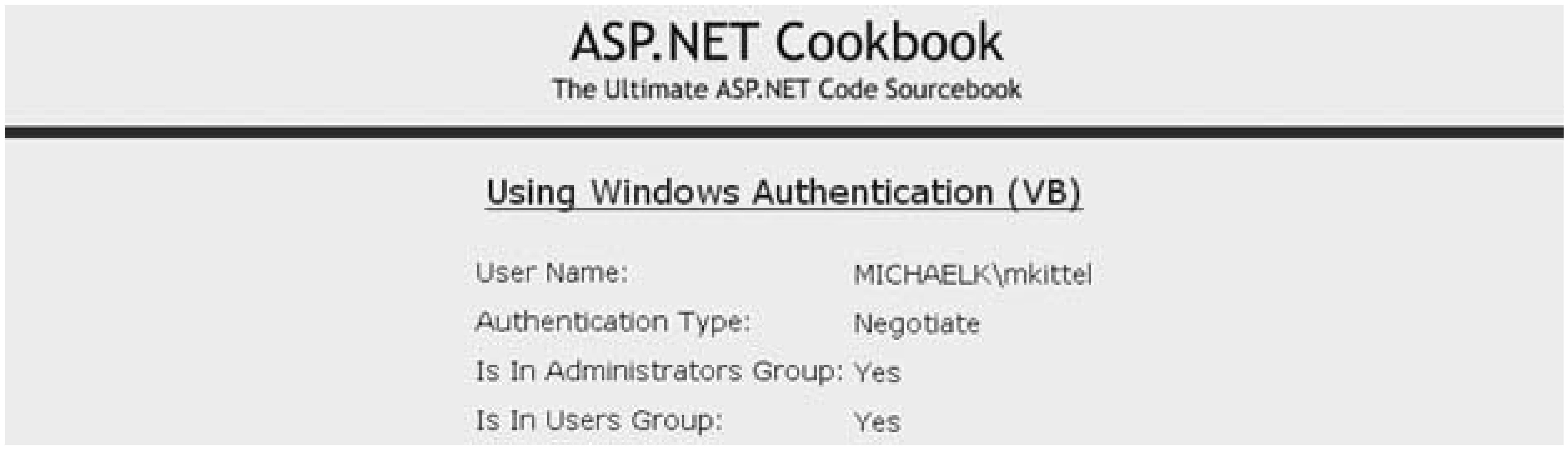
User Name

Password

Domain

☐ Save this password in your password list

Figure 9-3. Windows authentication sample page


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Using Windows Authentication (VB)

User Name:	MICHAELK\mkittel
Authentication Type:	Negotiate
Is In Administrators Group:	Yes
Is In Users Group:	Yes

Discussion

Windows authentication is a useful means of authenticating users of web applications that run on an intranet. Windows authentication allows you to assume that each user has a valid Windows account with appropriate permissions for accessing the network resources. This is an advantage to you as a web application developer because it saves you from having to maintain all this information separately in your application.

The setup required for using Windows authentication is similar to the setup performed for Forms authentication. The big difference is the role IIS plays in the authentication. To support Forms authentication, IIS is configured to allow anonymous access. In other words, IIS does not perform any authentication, leaving the task of authenticating and authorizing users to ASP.NET. (See Recipe 9.1 for more on Forms authentication.)

For Windows authentication, IIS must be configured to block anonymous access and must be configured to use Windows integrated authentication or basic authentication. We recommend Windows integrated authentication because this method does not send the user password over the network in clear text. With Windows authentication, IIS verifies that the user is allowed to access the application; then ASP.NET performs the authorization for the requested resource. The operating system can also be involved in the authorization by using Access Control Lists (ACLs) to limit access to resources by specific users.

After you set up IIS, the *web.config* file should be set up with the authentication mode set to `Windows`:

```
<authentication mode="Windows" />
```

The `<identity>` element should be set to `impersonate`:

```
<identity impersonate="true" userName="" password="" />
```

This configures ASP.NET to impersonate the user authenticated by IIS for all resource requests when the userName and password are empty strings. If you want all requests to use a different account than IIS used for authentication, the userName and password attributes of the <identity> element can be set to the desired username and password. However, there are two negatives if you do this. First, th password for the account is in clear text in *web.config*, which can cause security risks. Second, logging and auditing cannot be done on a per-user basis.

The <authorization> section is configured to deny access to all users:

```
<authorization>
 <deny users="*" /> <! Deny all users >
</authorization>
```

This is done because <location> elements will be added to define the authorizations for each page.

To control the access to each page, add a <location> element. This provides the maximum flexibility in controlling access to each page in your application. When using Windows authentication, roles are synonymous with groups. Therefore, the <allow> element should contain the list of groups (roles) allowed to access the given page. The <deny users="*"/> element should always be provided after the <allow> element to deny access to all users not listed in the previous roles. For example:

```
<location path="DisplayUserInformation.aspx">
   <system.web>
 <authorization>
    <allow roles="BuiltIn\Users,
        BuiltIn\Administrators"/>
    <deny users="*"/>
 </authorization>
   </system.web>
</location>
```

Group (role) names must be fully qualified. When using local built-in groups such as Users and Administrators, the fully qualified names are BuiltIn\Users and BuiltIn\Administrators. When using groups you have created, you must include the computer name, such as <*MyComputer*> \Testers. When using domain groups, you must include the domain name, such as <*DomainName*>\Testers.

As described in Recipe 9.2, you can place pages with the same access requirements in folders and include a <location> element defining the access to the folders. See Recipes 9.2 and 9.3 for more information on using folders in this way, including a discussion of the pros and cons of various folder-related approaches.

No other code is required in your application to implement Windows authentication.

You can access the user credentials in your application by using the identity property from the current context. Because Windows authentication is being used and more information is available for the user than is available using Forms authentication, the`identity` property should be cast as a `WindowsIdentity` type to access these additional properties:

VB

```
 identity = CType(Context.User.Identity, WindowsIdentity)
```

C#

```
 identity = (WindowsIdentity)(Context.User.Identity);
```

Windows authentication, the client browser, IIS, and Windows perform many functions behind the scenes. If you access the application from the same machine or from a machine in the same domain, you may not be prompted to enter your username and password. This is caused by the browser automatically sending your credentials when the challenge is issued by IIS. Whether or not this happens is a function of the requested URL, how IIS is configured, and how your browser is configured. The details of this configuration are beyond the scope of this book. If you're interested in this topic, consult your network administrator, who will probably know all the fine points.

See Also

Recipes 9.1, 9.2, and 9.3; MSDN documentation for IIS setup (search for "IIS authentication")

Example 9-11. web.config for Windows authentication

```
<?xml version="1.0"?>
<configuration>
 <system.web>

    …

   <authentication mode="Windows" />
   <identity impersonate="true" />
   <authorization>
     <deny users="*" />
     <!-- Deny all users -->
   </authorization>

     …

   </system.web>
```

```
  <!--
  ****************************************************************************
  The following section defines the pages in the application and the
  roles (groups) that are allowed to access them. Any group defined
  in Windows can be used. NOTE: The groups must be the fully
  qualified names such as BuiltIn\Administrators, etc.
  ****************************************************************************
  -->
  <location path="DisplayUserInformation.aspx">
    <system.web>
      <authorization>

   <allow roles="BuiltIn\Users,
     BuiltIn\Administrators"/>
   <deny users="*"/>
    </authorization>
  </system.web>
 </location>
</configuration>
```

Example 9-12. Windows authentication sample page (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="DisplayUserInformation.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.DisplayUserInformation"
 Title="Display User Information" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Using Windows Authentication (VB)
 </div>
 <table width="60%" align="center" border="0">
   <tr>
  <td class="labelText">User Name: </td>
  <td>
    <asp:Label ID="txtUserName" Runat="server"
     CssClass="labelText" />
  </td>
   </tr>
   <tr>
  <td class="labelText">Authentication Type: </td>
  <td>
    <asp:Label ID="txtAuthenticationType" Runat="server"
    CssClass="labelText" />
  </td>
   </tr>
   <tr>
  <td class="labelText">Is In Administrators Group: </td>
  <td>
    <asp:Label ID="txtAdminGroup" Runat="server"
    CssClass="labelText" />
  </td>
   </tr>
   <tr>
  <td class="labelText">Is In Users Group: </td>
  <td>
    <asp:Label ID="txtUsersGroup" Runat="server"
    CssClass="labelText" />
  </td>
   </tr>
 </table>
</asp:Content>
```

Example 9-13. Windows authentication sample page code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System.Security.Principal

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' DisplayUserInformation.aspx
 ''' </summary>
 Partial Class DisplayUserInformation
  Inherits System.Web.UI.Page
  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
  Dim identity As WindowsIdentity

  'get the current user's identity
  identity = CType(Context.User.Identity, WindowsIdentity)

  'output the user's name and authentication type
  txtUserName.Text = identity.Name
  txtAuthenticationType.Text = identity.AuthenticationType

  'check to see if the user is a member of the administators group
  If (Context.User.IsInRole("BuiltIn\Administrators")) Then
   txtAdminGroup.Text = "Yes"
  Else
   txtAdminGroup.Text = "No"
  End If

  'check to see if the user is a member of the users group
  If (Context.User.IsInRole("BuiltIn\Users")) Then
   txtUsersGroup.Text = "Yes"
  Else
   txtUsersGroup.Text = "No"
  End If
 End Sub 'Page_Load
  End Class 'DisplayUserInformation
End Namespace
```

Example 9-14. Windows authentication sample page code-behind (.cs)

```
using System;
using System.Security.Principal;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 /// DisplayUserInformation.aspx
 /// </summary>
 public partial class DisplayUserInformation : System.Web.UI.Page
 {
   ///***********************************************************************
   /// <summary>
   /// This routine provides the event handler for the page load event.
   /// It is responsible for initializing the controls on the page.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void Page_Load(object sender, EventArgs e)
   {
  WindowsIdentity identity = null;

  // get the current user's identity
  identity = (WindowsIdentity)(Context.User.Identity);
  // output the user's name and authentication type
  txtUserName.Text = identity.Name;
  txtAuthenticationType.Text = identity.AuthenticationType;

  // check to see if the user is a member of the administators group
  if (Context.User.IsInRole("BuiltIn\\Administrators"))
  {
   txtAdminGroup.Text = "Yes";
  }
  else
  {
   txtAdminGroup.Text = "No";
  }

  // check to see if the user is a member of the users group
  if (Context.User.IsInRole("BuiltIn\\Users"))
  {
   txtUsersGroup.Text = "Yes";
  }
  else
  {
   txtUsersGroup.Text = "No";
  }
   } // Page_Load
 } // DisplayUserInformation
}
```

PREV

Recipe 9.6. Using Membership and Roles

Problem

You want to secure your web site by writing only a minimum amount of code.

Solution

Use ASP.NET 2.0's Membership and Role providers. The solution involves the following steps:

Modify *web.config* as follows:

1. Add the `<authentication>, <authorization>` , and `<location>` elements, as described in Recipe 9.3.
2. Add a `<membership>` element with a `<provider>` element defining the provider used to authenticate users.
3. Add a `<roleManager>` element with a `<provider>` element defining the provider used to manage the roles for users of your application.

In the *.aspx* file for the login page:

1. Add a `Login` control.
2. Customize the `Login` control as required by your application.

In the *.aspx* file for pages in your application, optionally add an `<asp:LoginName>` control to display the logged in user's name and an `<asp:LoginStatus>` control to provide the ability to log out.

The code we've created to illustrate this solution is shown in Examples 9-15, 9-16 through 9-17 . Example 9-15 shows the modifications we make to *web.config* to use the Membership and Role providers. Example 9-16 shows the *.aspx* file for the login page, and Example 9-17 shows the *.aspx* file for a page that displays the user's name and provides the ability to log out of the application.

Discussion

ASP.NET 1.x simplified the coding required to control access to pages in your application. The infrastructure to handle authentication and authorization for pages in your application requires no code in the individual pages; however, it does require you to validate the user's credentials, store the user information in a cookie, and extract the information from the cookie for each page request. Though not much code was required in ASP.NET 1.x (see Recipe 9.3), authentication and authorization can be implemented without writing any code whatsoever in ASP.NET 2.0.

Two providers are available in ASP.NET 2.0 to handle authentication and authorization. The Membership provider handles the validation of user credentials and the Role provider handles the user roles to support authorization. In addition, a configuration tool is provided to support managing users and their roles.

In our example application we have modified the *web.config* file to include the `<authentication>`, `<authorization>` , and `<location>` elements, as described in Recipe 9.3. These elements are used to define the type of authentication used and the roles required to access the pages in the application.

To take advantage of the membership functionality, a `<membership>` element is added to *web.config* . This configures the provider that will be used to validate the user's credentials when she logs in.

```
<membership defaultProvider="AspNetSqlMembershipProvider"
  userIsOnlineTimeWindow="15">
<providers>
  <remove name="AspNetSqlMembershipProvider" />
  <add name="AspNetSqlMembershipProvider"
  type="System.Web.Security.SqlMembershipProvider"
   connectionStringName="LocalSqlServer"
   enablePasswordRetrieval="false"
  enablePasswordReset="true"
  requiresQuestionAndAnswer="true"
  applicationName="/"
  requiresUniqueEmail="false"
   minRequiredPasswordLength="1"
   minRequiredNonalphanumericCharacters="0"
  passwordFormat="Hashed"
   maxInvalidPasswordAttempts="5"
   passwordAttemptWindow="10"
  passwordStrengthRegularExpression="" />
  </providers>
</membership>
```

The `defaultProvider` attribute of the membership element defines the provider that will be used for managing users. The `userIsOnlineTimeWindow` attribute specifies the number of minutes after the last access the user is still considered to be online. After the specified number of minutes elapses with no activity from the user, the user is logged out. The default is 15 minutes.

```
<membership defaultProvider="AspNetSqlMembershipProvider"
  userIsOnlineTimeWindow="15">

…
</membership>
```

The `defaultProvider` attribute should always be explicitly set. Omitting the `defaultProvider` attribute will implicitly define the membership provider as the first provider listed within the `<membership>` element. This could result in unexpected behavior of your application if the list of providers is modified.

In addition, you should always define the provider(s) required for your application since a provider is defined in `machine.config` . If your application does not define its own provider(s), the provider defined in `machine.config` will be used; this may cause your application to behave unexpectedly.

The `<membership>` element contains a `<providers>` element with one or more `<add>` elements to define the provider(s) available for managing users. The `name` and `type` attributes of the `<add>` element are required. All other attributes are optional and are a function of the individual providers.

In our example we are using the `SqlMembershipProvider` , which supports multiple attributes to configure the provider. Table 9-1 defines the attributes available for the `SqlMembershipProvider` .

Table 9-1. SqlMembershipProvider attributes

Attribute	Description
`name`	Defines the name for the provider. Any value can be used.
`type`	Specifies the class used to provide the functionality for the provider. Must be set to `System.Web. Security.SqlMembershipProvider` for the `SqlMembershipProvider` .
`connectionStringName`	Specifies the name of the connection string used to access the SQL database used by the provider. Must be set to one of the connection strings defined in the `<connectionStrings>` element of *web.config* . Required for the `SqlMembershipProvider` .
`enablePasswordRetrieval`	Specifies if retrieval of passwords is supported. Valid values are `true` and `false` . The default value is `false` .
`enablePasswordReset`	Specifies if resetting passwords is supported. Valid values are `true` and `false` . The default value is `true` .
`requiresQuestionAndAnswer`	Specifies if a user must provide an answer to his security question when retrieving his password. Valid values are `true` and `false` . The default value is `true` .
`applicationName`	Specifies the name of the application using the membership provider. The default value is the value of the `System.Web.HttpRequest.ApplicationPath` property.
`requiresUniqueEmail`	Specifies if the membership provider requires user email addresses to be unique. Valid values are `true` and `false` . The default value is `true` .

Attribute	Description
`minRequiredPasswordLength`	Specifies the minimum length of passwords allowed by the member ship provider. The default value is `7` .
`minRequiredNonalphanumericCharacters`	Specifies the minimum number of nonalphanumeric characters (spaces, punctuation, etc.) that must be included in a password. The default value is `1` .
`passwordFormat`	Specifies the format in which passwords are stored in the database used by the membership provider. Valid values are `Clear, Encrypted` , and `Hashed. Clear` specifies that passwords are to be stored in plain text. `Encrypted` specifies that passwords are to be encrypted. `Encrypted` should be used when password values must be retrievable. `Hashed` specifies that passwords are to be stored using a one-way hash algorithm. Passwords stored in hashed format cannot be retrieved. The default is `Hashed` .
`maxInvalidPasswordAttempts`	Specifies the maximum number of attempts that can be made before the user's account is locked out. The default value is `5` .
`passwordAttemptWindow`	Specifies the elapsed number of minutes during which the maximum number of invalid password attempts is allowed. The default value is `10` .
`passwordStrengthRegularExpression`	Specifies a regular expression to be used to validate the strength of user passwords. The default is an empty string.

In our application, we have set the `minRequiredNonalphanumericCharacters` to `0` to simplify the use of the example. Evaluate the security needs of your application and set the `minRequiredPasswordLength, minRequiredNonalphanumericCharacters, maxInvalidPasswordAttempts` , and `passwordAttemptWindow` attributes accordingly.

To take advantage of the role management functionality provided by the Role Manager, a `<roleManager>` element is added to *web.config* . This configures the provider used to associate users with roles and support authorization as a function of the user's role.

```
<roleManager defaultProvider="AspNetSqlRoleProvider"
   enabled="true"
   cacheRolesInCookie="true"
     cookieName=".ASPNETCookbookVBSecurity951Roles"
    cookieTimeout="30"
   cookiePath="/"
   cookieRequireSSL="false"
   cookieSlidingExpiration="true"
   cookieProtection="All" >

   <providers>
  <remove name="AspNetSqlRoleProvider" />
  <add name="AspNetSqlRoleProvider"
   type="System.Web.Security.SqlRoleProvider"
   connectionStringName="LocalSqlServer"
   applicationName="/" />
  </providers>
   </roleManager>
```

The `< roleManager>` element has multiple attributes used to configure the role manager. Table 9-2 defines the available attributes for the `<roleManager>` element.

Table 9-2. RoleManager attributes

Attribute	Description
`defaultProvider`	Specifies the provider that will be used for role management. Should always be explicitly set to the desired provider. The default, if not specified, is the first provider listed within the `<roleManager>` element.
`enabled`	Specifies if role management is being used. Valid values are `true` and `false` . The default value is `false` .
`cacheRolesInCookie`	Specifies if user role information is cached in a cookie. Valid values are `true` and `false` . The default value is `true` .
`cookieName`	Specifies the name of the cookie used to cache the user roles. The default value is `.ASPXROLES` .
`cookieTimeout`	Specifies the time in minutes before the cookie expires. The default value is 30 minutes.
`cookiePath`	Specifies the path of the cookie used for storing role information. Most browsers treat the path as case-sensitive and will not return the cookie for a request that does not match the value provided for the path attribute. The result will be having the users redirected as if they were not logged in, since the role data will not be available. Unless your application requires specifying the path, we recommend that you leave the path as "/".
`cookieRequireSSL`	Specifies if SSL is required. Valid values are `TRue` and `false` . The default value is `false` .
`cookieSlidingExpiration`	Specifies if sliding expiration of the role cookie is enabled. If set to `true` , the roles cookie time to expire will be refreshed on every page request. If set to `false` , the roles cookie will expire at the time set when the cookie was created. The default value is `true` .

Attribute	Description
cookieProtection	Specifies the protection method used for the roles cookie. Possible values are All, None, Encryption , and Validation. Validation specifies that the cookie data will be validated to ensure it was not altered in transit. Encryption specifies that the cookie is encrypted. All specifies that data validation and encryption will be used.None specifies that no protection will be provided for the cookie information. The default is All and is highly recommended because it offers the highest level of protection for this authentication cookie.

The cookieName specified in the <roleManager> element must be different from the value specified for the name attribute in the <forms> element. The name attribute of the <forms> element specifies the name of the authentication cookie. The cookieName attribute of the <roleManager> element specifies the name of the cookie used to store user role data. If both are set to the same value, the user will be properly redirected to the first page after he is authenticated. Attempts to access additional pages in the application will result in the user being redirected to the login page. This is because theroles data overwrites the authentication data in the cookie, thus deleting the data ASP.NET uses to determine if a user is authenticated.

One or more <provider> elements can be specified to define the providers available to manage user roles. In our application we are using the SqlRoleProvider . As with most providers, attributes of the element are used to configure the provider. Table 9-3defines the attributes available for the SqlRoleProvider .

Table 9-3. SqlRoleProvider attributes

Attribute	Description
name	Defines the name for the provider. Any value can be used.
type	Specifies the class used to provide the functionality for the provider. Must be set to System. Web. Security.SqlRoleProvider for the SqlRoleProvider .
connectionStringName	Specifies the name of the connection string used to access the SQL database used by the provider. Must be set to one of the connection strings defined in the <connectionStrings> element of *web.config* . Required for the SqlRoleProvider .
applicationName	Specifies the name of the application using the membership provider. The default value is the value of the System.Web.HttpRequest.ApplicationPath property.

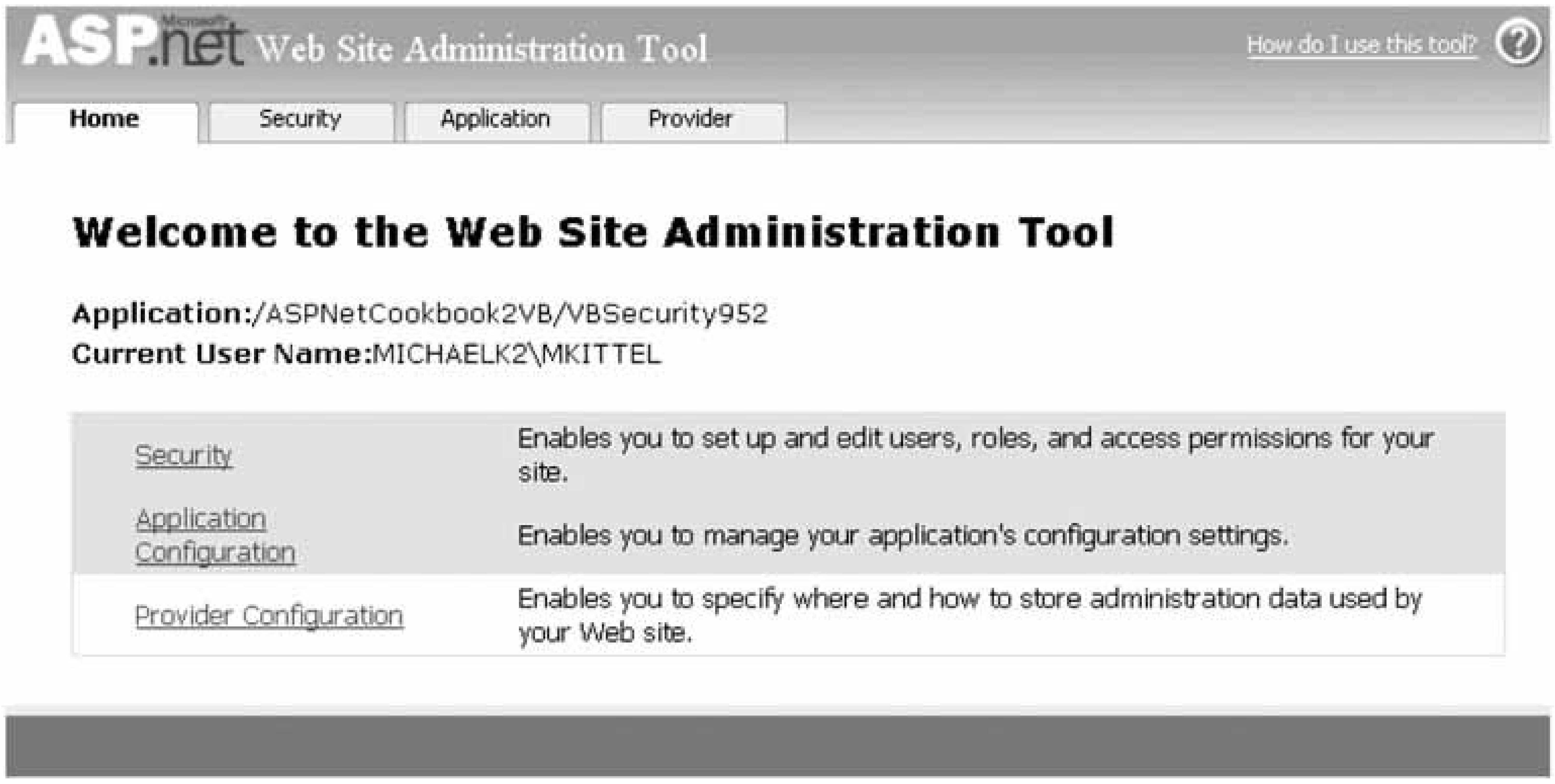
In our application, we are using SQLExpress as the data store for the user information. ASP.NET 2.0 makes creation of the database required to support membership and roles simple. If you are using

Visual Studio 2005, you can select your website in the Solution Explorer and then select the WebSite → ASP.NET Configuration toolbar item. This will launch the Web Site Administration Tool as shown i Figure 9-4 .

To manage the users and roles, select the Security tab. The security page will be displayed, as showr in Figure 9-5 , with options to configure the security aspects of your application. We have configured everything required except for the creation of the users and roles.

To manage the user roles, select the Create or Manage Roles link. The roles administration page will be displayed, as shown in Figure 9-6. Add the roles required for your application. For our application, we have added a `User` and an `Admin` role.

Figure 9-4. Web Site Administration Tool home page

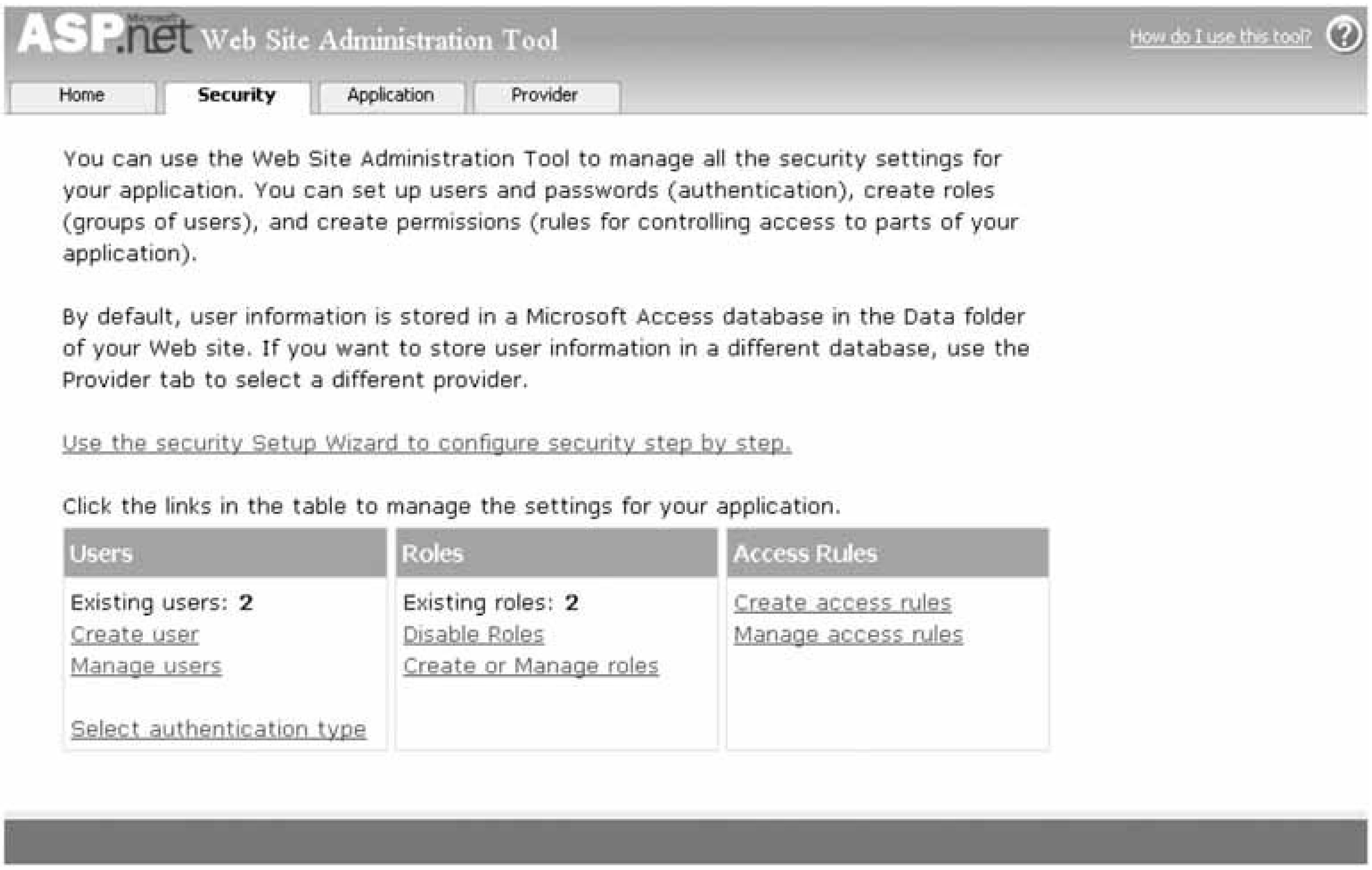


Welcome to the Web Site Administration Tool

Application:/ASPNetCookbook2VB/VBSecurity952
Current User Name:MICHAELK2\MKITTEL

Security	Enables you to set up and edit users, roles, and access permissions for your site.
Application Configuration	Enables you to manage your application's configuration settings.
Provider Configuration	Enables you to specify where and how to store administration data used by your Web site.

Figure 9-5. Web Site Administration Tool security page



You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

By default, user information is stored in a Microsoft Access database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

Use the security Setup Wizard to configure security step by step.

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: **2**	Existing roles: **2**	Create access rules
Create user	Disable Roles	Manage access rules
Manage users	Create or Manage roles	
Select authentication type		

Figure 9-6. Web Site Administration Tool role management page

When the required roles have been added, click the back button on the Role Management page and select the Manage Users link when the security home page is displayed to manage the users in your application. The user management page will be displayed, as shown in Figure 9-7. Add the users and assign the roles for each user required for your application. For our application, we have created two users: "user" and "admin."

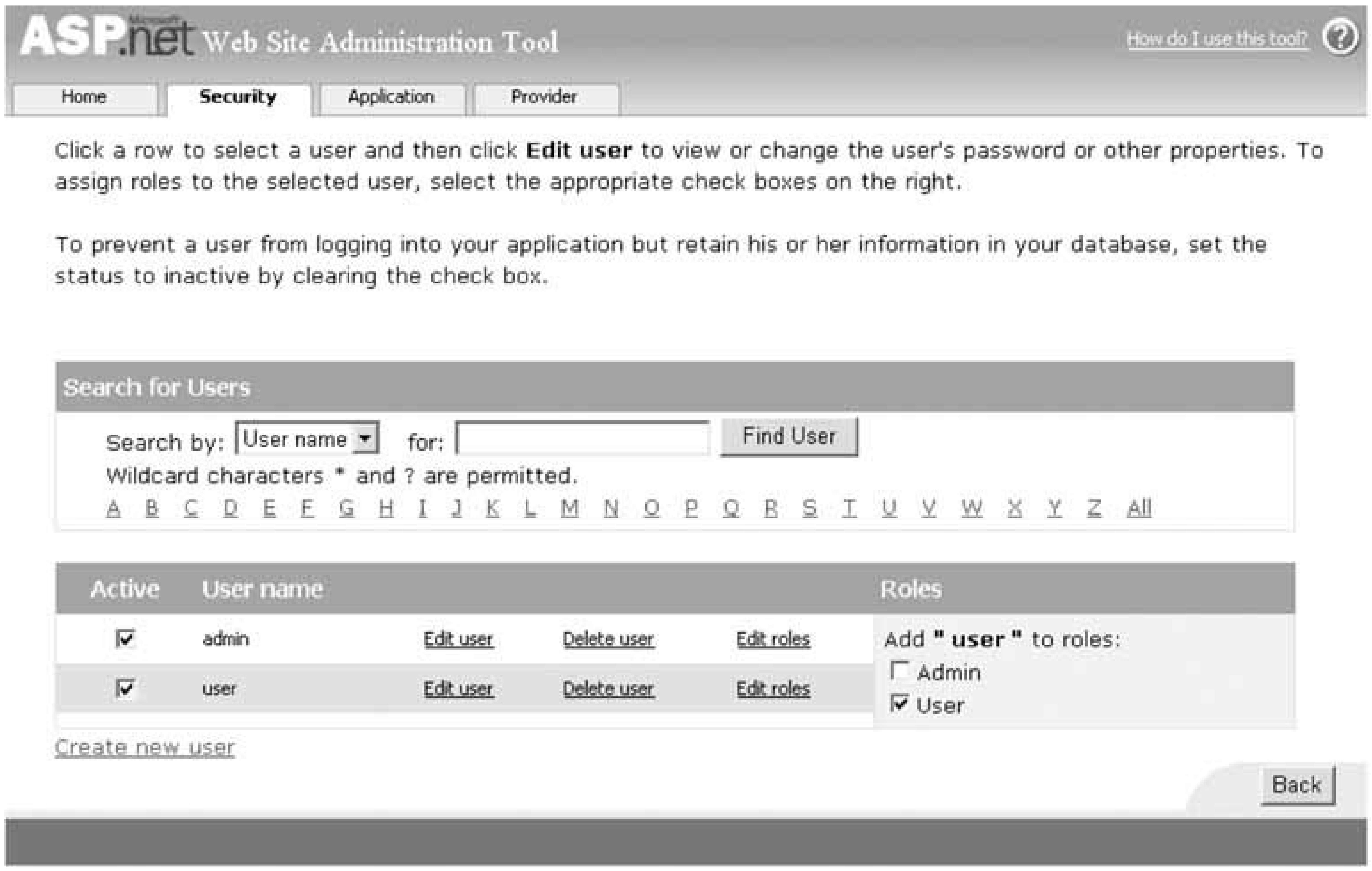
When you use the WebSite Administration Tool, a `SQLExpress` database is created in the `App_Data` folder of your application. The database is contained in a file named *ASPNETDB.MDF* .

Now that the application is configured and the data store is created, you will need to create a login page that uses an `asp:Login` control. The only attributes of the `asp:Login` control that must be set are the `ID` and `runat` attributes; however, the control provides a multitude of attributes to allow you to modify the appearance of the rendered control to meet the needs of your application.

```
<asp:Login ID="Login1" runat="server"  />
```

The `asp:Login` control will automatically make the appropriate calls to validate the users credentials, create the authentication and roles cookies, and redirect the user to the page specified in the `defaultUrl` attribute of the `<forms>` element in the *web.config* file. No code is required on your part.

Figure 9-7. Web Site Administration Tool user management page



Click a row to select a user and then click **Edit user** to view or change the user's password or other properties. To assign roles to the selected user, select the appropriate check boxes on the right.

To prevent a user from logging into your application but retain his or her information in your database, set the status to inactive by clearing the check box.

Wildcard characters * and ? are permitted.

Active	User name				Roles
✓	admin	Edit user	Delete user	Edit roles	Add " **user** " to roles:
✓	user	Edit user	Delete user	Edit roles	☐ Admin ✓ User

In our application, we have added an `asp:LoginName` and an `asp:LoginStatus` control on our secured pages, as shown in Example 9-17. The `asp:LoginName` control provides the ability to display the name of the logged-in user and the `asp:LoginStatus` control provides a link for the user to log out. When the user clicks the "log out" link of the `asp:LoginStatus` control, the control automatically makes the appropriate calls to log the user out and take the action defined by the `LogoutAction` attribute. In our application, we have set the action to `RedirectToLoginPage` , which will redirect the user to the page specified in the `loginUrl` attribute of the `<forms>` element in the *web.config* file.

> If you want to use controls other than the `asp:Login` and `asp:LoginStatus` controls to handle the login and logout actions from the user, you can use the Membership and Role providers. The `Membership` class provides a full set of methods to manage the users of your application programmatically. The `Validate` method accepts the user's credentials and returns a Boolean value indicating whether the credentials are valid or not. Combining this with the login code described in Recipes 9.1, 9.2, and 9.3, you can use the functionality of the Membership and Role providers but keep control over the login and logout process. For more information on the available methods, see the documentation in the MSDN Library for the Membership and Roles classes.

See Also

Recipes 9.1, 9.2, and 9.3; MSDN Library for additional information on the Membership and Roles classes

Using SQL Server Instead of SQLExpress with the Membership and Role Providers

SQL Server can be used with the Membership and Role providers. The SQL Server database will not be automatically created for you, but Microsoft has provided an application to assist you. The application provides the ability to create a new database with the table structure required by the Membership and Role providers or add the tables to an existing database. The application is *aspnet_regsql.exe* and by default is located in the *C:\WINNT\Microsoft.NET\Framework\<version>* folder, where *<version>* is the version number of the framework installed on your machine.

After creating the database, you will need to add a connection string to the *web.config* file that defines the name, location, and login credentials for the Membership database. The [*server*], [*user*], and [*password*] settings shown below should be set accordingly for your server and database:

```
<connectionStrings>
  <remove name="sqlConnectionString" />
  <add name="sqlConnectionString"
 connectionString="Data Source=[server];
 Initial Catalog=aspnetdb;
 UID=[user];
 PWD=[password];
 persist security info=False;
 Connection Timeout=30;"/>
</connectionStrings>
```

The final step for configuring SQL Server for the Membership and Role providers is to set the `connectionStringName` attribute of the providers to the name of the connection string added for SQL Server:

```
connectionStringName="sqlConnectionString"
```

Example 9-15. Changes to web.config to use Membership and Role providers

```
<?xml version="1.0"?>
```

```
<!--
 Note: As an alternative to hand editing this file you can use the
 web admin tool to configure settings for your application. Use
 the Website->Asp.Net Configuration option in Visual Studio.
 A full list of settings and comments can be found in
 machine.config.comments usually located in
 \Windows\Microsoft.Net\Framework\v2.x\Config
-->
<configuration>

 <appSettings/>
 <connectionStrings>
   <remove name="LocalSqlServer" />
   <add name="LocalSqlServer"
   connectionString="data source=.\SQLEXPRESS;
   Integrated Security=SSPI;
   AttachDBFilename=|DataDirectory|aspnetdb.mdf;
   User Instance=true"
   providerName="System.Data.SqlClient" />
  </connectionStrings>

  <system.web>

    …

  <authentication mode="Forms" >
  <forms name=".ASPNETCookbookVBSecurity951"
     loginUrl="Login.aspx"
     protection="All"
     timeout="30"
     path="/"
     defaultUrl="Home.aspx" />
 </authentication>

 <authorization>
  <deny users="*" /> <!-- Deny all users -->
 </authorization>

 <membership defaultProvider="AspNetSqlMembershipProvider"
    userIsOnlineTimeWindow="15">
  <providers>
    <remove name="AspNetSqlMembershipProvider" />
    <add name="AspNetSqlMembershipProvider"
   type="System.Web.Security.SqlMembershipProvider"
   connectionStringName="LocalSqlServer"
   enablePasswordRetrieval="false"
   enablePasswordReset="true"
   requiresQuestionAndAnswer="true"
   applicationName="/"
   requiresUniqueEmail="false"
   minRequiredPasswordLength="4"
   minRequiredNonalphanumericCharacters="0"
```

```
    passwordFormat="Hashed"
    maxInvalidPasswordAttempts="5"
    passwordAttemptWindow="10"
    passwordStrengthRegularExpression="" />
   </providers>
  </membership>

  <roleManager defaultProvider="AspNetSqlRoleProvider"
     enabled="true"
     cacheRolesInCookie="true"

     cookieName=".ASPNETCookbookVBSecurity951 Roles"
     cookieTimeout="30"
     cookiePath="/"
     cookieRequireSSL="false"
     cookieSlidingExpiration="true"
     cookieProtection="All" >
   <providers>
     <remove name="AspNetSqlRoleProvider" />
     <add name="AspNetSqlRoleProvider"
     type="System.Web.Security.SqlRoleProvider"
     connectionStringName="LocalSqlServer"
     applicationName="/" />
   </providers>
  </roleManager>

  …

</system.web>
<!--
*****************************************************************************
  The following section provides public access to pages that do not
  require authentication. An entry must be included for each page
  or folder that does not require authentication.
*****************************************************************************
-->
   <location path="PublicPage.aspx">
     <system.web>
    <authorization>
      <allow users="*"/>
    </authorization>
  </system.web>
   </location>

<!--
*****************************************************************************
  The following section defines the pages that require authentication
  for access. An entry must be included for each page or folder that
  requires authentication with a list of the roles required for access
  to the page.

  Valid Roles are as follows.
```

```
 NOTE: The roles must be entered exactly as listed.

   User
   Admin
*****************************************************************************
-->
<location path="Home.aspx">
  <system.web>
    <authorization>
      <allow roles="User,
      Admin"/>

  <deny users="*"/>
  </authorization>
   </system.web>
 </location>

 <location path="AdminPage.aspx">
   <system.web>
     <authorization>
    <allow roles="Admin"/>
    <deny users="*"/>
  </authorization>
   </system.web>
 </location>

</configuration>
```

Example 9-16. Login page (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="Login.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.Login"
 Title="Login" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
   Using Membership and Roles With SQLExpress (VB)
 </div>
 <div align="center">
   <asp:Login ID="Login1" runat="server"
    TitleText=""
    UserNameLabelText="Login ID: "
    PasswordLabelText="Password: "
    DisplayRememberMe="true"
    RememberMeText="Remember Me"
    LabelStyle-CssClass="labelText"
    CheckBoxStyle-CssClass="labelText"
    TextBoxStyle-CssClass="labelText" >
  <CheckBoxStyle CssClass="labelText" />
  <TextBoxStyle CssClass="labelText" />
  <LabelStyle CssClass="labelText" />
   </asp:Login>
   <div align="center" class="labelText">
     <br />Login ID=user, Password=user for access by member of User role
     <br />Login ID=admin, Password=admin for access by member of Admin role
   </div>
   <br />
   <input type="button" value="Attempt Access without Login"
    onclick="document.location='Home.aspx'" />
   <br /><br />
   <input type="button" value="Access Public Page"
    onclick="document.location='PublicPage.aspx'" />
 </div>
</asp:Content>
```

Example 9-17. Home page (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="Home.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.Home"
 Title="Home" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
   Using Membership and Roles With SQLExpress (VB)
 </div>
 <table width="60%" align="center" border="0">
   <tr>
     <td align="center">
    <asp:LoginName ID="LoginName1" runat="server" />
   </td>
   </tr>
   <tr>
     <td align="center">
    <br />
    <asp:LoginStatus ID="LoginStatus1" runat="server"
     LogoutText="Log Out"
     LogoutAction="RedirectToLoginPage" />
  </td>
   </tr>
 </table>
</asp:Content>
```

PREV

PREV

Chapter 10. Profiles and Themes

PREV

PREV

10.0 Introduction

If you were interested in building profiles and themes into your own application in ASP.NET 1.x, you basically had to create your own. For instance, you had to provide your own mechanisms for storing and retrieving user profile information and for allowing users to choose the look and feel they wantec for your application. The latter was sufficiently daunting that beginning or mid-level developers rarely attempted it.

Fortunately, with ASP.NET 2.0's profile support, the days of having to create your own are gone. For instance, in ASP.NET 2.0, you can include user profile data in your application without writing any code to persist the data. You can inherit a profile, which is handy when you want to use the same profile definition in multiple applications. You can store profile information for users who are not logged into your application, and you can create a mechanism to periodically remove user profile dat that is no longer being used. All of these scenarios are illustrated by the recipes in this chapter.

Themes are another new feature in ASP.NET 2.0 and provide a browser-independent way of "skinning" a set of controls. In ASP.NET 2.0,*themes* consist of collections of (Cascading Style Sheets) CSS files, *.skin* files, and images that contain a definition for each server control you use in your application and images. If you want to provide users the ability to choose the look and feel they wan for your application, you can store their selected theme in their profile and set the selected theme in the pre-initialization code of the pages in your application. Recipe 10.5 shows you how to create an ASP.NET 2.0 theme, and Recipe 10.6 shows you how to manage user-personalized themes.

Web parts are another new ASP.NET 2.0 feature that we could have covered in this chapter. Because the topic is sufficiently rich in its own right, however, we decided to cover it in a separate chapter all its own, which follows this one.

Before delving into this chapter, you should know that Microsoft has clarified its personalization and profile terminology in ASP.NET 2.0:

Membership

Describes the authentication features

Role Manager

Describes the authorization features as a function of a user's roles

Profile

Describes the features used to store information about a user

Personalization

Describes the personalization that can be done with web parts

All of these topics, with the exception of personalization, are covered in one way or another in the various recipes in this chapter. See Chapter 11 for more information on personalization.

PREV

Recipe 10.2. Using Profiles

Problem

You want to make use of user profile data throughout your application without having to implement the infrastructure yourself.

Solution

Use the ASP.NET 2.0 membership and profile features by implementing the membership features described in Recipe 9.5, modify *web.config* to enable user profiles, define in *web.config* the specific user data to include in the profile, provide the ability to update the user profile data, and use the dat as required in your application.

Modify *web.config* as follows:

1. Add a `<profile>` element with a `<provider>` element defining the provider used to store and retrieve user-specific data.
2. Add a `<properties>` element and an `<add>` element for each user-specific data item to be included in the profile.

In the *.aspx* file for the page used to enter the user-specific data:

1. Add a server control for each user-specific data item.
2. Add an Update button (or equivalent) to initiate the updating of the user-specific data.

In the code-behind class for the page used to enter user-specific data, use the .NET language of your choice to:

1. Initialize the controls in the *.aspx* file with the current data from the user's profile.
2. Implement an event handler for the Update button click event and update the user's profile from the data entered by the user.

The solution we have implemented to demonstrate the solution is shown in Examples 10-1, 10-2, 10-3 through 10-4. Example 10-1 shows the modifications made to *web.config*. Example 10-2 shows the *.aspx* file used to enter the user-specific data, and Examples 10-3 (VB) and 10-4 (C#) show the code-behind classes for the page used to enter the user-specific data.

Discussion

With ASP.NET 1.x, as well as with classic ASP, no functionality was provided to maintain user profile information. If you needed the ability to maintain user profile information, you had to implement you own solution, which typically required a significant effort. With ASP.NET 2.0, the ability to maintain user profile information is provided by using the Profile features, and, in most cases, the only code required in your application is to use the information and to provide the ability for users to update their information.

Maintaining user profile information requires defining the data to be maintained, having the ability to store and retrieve the data, and having the ability to access the data. ASP.NET 2.0 simplifies each of these tasks.

The simplest way to define the data to include in the user profile is using*web.config*. By adding a `<properties>` element and `<add>` elements to the `<profile>` element for each user profile item to be stored, ASP.NET 2.0 dynamically creates a class containing the profile data. If you are using Visual Studio 2005, the class will be created in the background and its contents will be available using `Intellisense`, making use of the profile data even easier.

```
<profile enabled="true"
   defaultProvider="AspNetSqlProfileProvider"
   automaticSaveEnabled="false" >

  …

 <properties>
   <add name="FirstName"
     type="System.String"
     serializeAs="String"

     allowAnonymous="false"
     defaultValue=""
     readOnly="false" />
   <add name="LastName"
     type="System.String"
     serializeAs="String"
     allowAnonymous="false"
     defaultValue=""
     readOnly="false" />

  …
  </properties>
</profile>
```

ASP.NET 2.0 includes one provider to handle the storage and retrieval of the user profile data, the

SqlProfileProvider. This provider supports storing the user profile information in SQL Express, SQL Server 7.0, SQL Server 2000, or SQL Server 2005 without requiring you to write any code. All that is required is adding a <providers> element to the <profile> element in *web.config*. The attributes available for the SqlProfileProvider are shown in Table 10-1.

```
<profile enabled="true"
   defaultProvider="AspNetSqlProfileProvider"
   automaticSaveEnabled="false" >

<providers>
    <remove name="AspNetSqlProfileProvider" />
 <add name="AspNetSqlProfileProvider"
   type="System.Web.Profile.SqlProfileProvider"
   connectionStringName="sqlConnectionString"
   applicationName="/" />
  </providers>

    …

</profile>
```

Table 10-1. SqlProfileProvider attributes

Attribute	Description
Name	Defines the name for the provider. Any value can be used.
Type	Specifies the class used to provide the functionality for the provider. Must be set to System.Web.Profile.SqlProfileProvider for the SqlProfileProvider.
connectionStringName	Specifies the name of the connection string used to access the SQL database used by the provider. Must be set to one of the connection strings defined in the <connectionStrings> element of *web.config*. Required for the SqlProfileProvider.
applicationName	Specifies the name of the application using the profile provider. The default value is the value of the System.Web.HttpRequest.ApplicationPath property.

The providers available to an application include all the providers defined in all the *.config* files in the application hierarchy. At a minimum, this will include *machine.config* and your application's *web.config*, but may also include additional *web.config* files if your application itself is hierarchical and has multiple *web.config* files. If a provider is included in multiple `config` files with the same name, an exception indicating that the provider exists will be thrown. To eliminate the possibility of accidental name collisions, include a `<remove>` element before the `<add>` element for the name of the provider you are adding. This is acceptable even if the provider does not exist since it is not an error to remove a nonexistent provider.

If you need to store the user profile information in a different data store, you can implement your own profile provider to store and retrieve the data. Refer to the `ProfileProvider` class in the MSDN Library for more information on implementing a custom profile provider.

In our application that implements this solution, we have implemented the `Membership` functionality for authentication and authorization, as described in Recipe 9.5, to provide the unique identification of the user for the profile data. The use of the `Membership` features in ASP.NET 2.0 is not required to use the `Profile` feature. All that is required is to provide a unique value for each user in the `User.Identity.Name` property of the `Principal` object used for authentication, which is used by the Profile Provider to identify the profile data required for the user. This means, for example, that the authentication solution provided in Recipe 9.1, which does not use `Membership`, can be used equally well.

The `Membership` and `Profile` providers share many tables in the database. If you want to use the `Profile` feature but do not intend to use the `Membership` features, you will or still need to add the tables required for the `Membership` and `Profile` providers. Refer to the "Using SQL Server Instead of SQLExpress with the Membership and Role Providers" sidebar in Recipe 9.5 for information on creating a database with the required tables or adding the required tables to an existing database.

The data to be included in the user profile is defined in *web.config*, as shown in Example 10-1. The definition of a data item (profile property) includes multiple attributes described in Table 10-2.

Table 10-2. Profile property attributes

Attribute	Description
`name`	Specifies the name of the property in the dynamically created Profile class. The name is required and must be unique.
`type`	Specifies the data type for the property. The type is required and can be any .NET data type or any custom data type of your choosing as long as the type can be serialized.

Attribute	Description
serializeAs	Specifies how the data is to be serialized for storage in the data store. Valid values are String, Xml, Binary, and ProviderSpecific. The default value is String.
allowAnonymous	Specifies if the property should be allowed for anonymous users. Valid values are true and false. The default value is false.
provider	Specifies the profile provider to be used to persist the property data. This provides the ability for individual properties to be persisted by different providers. The default value is the provider specified in thedefaultProvider attribute of the <provider> element.
defaultValue	Specifies the default value used when no profile data is available for the property. This value is not stored in the data store by default. It is used to initialize the property in the Profile object only.
readOnly	Specifies if the property should be read-only. Valid values areTRue and false. The default value is false.

The data for our user profile uses a <group> element to group the properties for an address into a MailingAddress group. This approach makes using the data in your code easier in cases where you have multiple similar groups, such as a mailing address and a shipping address.

```
 <profile enabled="true"
      defaultProvider="AspNetSqlProfileProvider"
    automaticSaveEnabled="false" >

    …

   <properties>
     <add name="FirstName"
    type="System.String"
    allowAnonymous="false"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
  <add name="LastName"
    type="System.String"
    serializeAs="String"
    allowAnonymous="false"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
  <group name="MailingAddress">
    <add name="Address1"
      type="System.String"
      serializeAs="String"
      allowAnonymous="false"
      provider="AspNetSqlProfileProvider"
      defaultValue=""
      readOnly="false" />
    <add name="Address2"
      type="System.String"
      serializeAs="String"
      allowAnonymous="false"
      provider="AspNetSqlProfileProvider"
      defaultValue=""
      readOnly="false" />
    <add name="City"
      type="System.String"
      serializeAs="String"
      allowAnonymous="false"
      provider="AspNetSqlProfileProvider"
      defaultValue=""
      readOnly="false" />
    <add name="State"
      type="System.String"
      serializeAs="String"
      allowAnonymous="false"
      provider="AspNetSqlProfileProvider"
      defaultValue=""
      readOnly="false" />
    <add name="ZipCode"
      type="System.String"
      serializeAs="String"
      allowAnonymous="false"
      provider="AspNetSqlProfileProvider"
      defaultValue=""
      readOnly="false" />
  </group>
  <add name="EmailAddresses"
    type="System.Collections.Specialized.StringCollection"
    serializeAs="Xml"
    allowAnonymous="false"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
   </properties>
 </profile>
```

To demonstrate how to handle collections of properties, the data in our application includes a collection of email addresses. The data type for this property is defined as`StringCollection`.

```
 <profile enabled="true"
     defaultProvider="AspNetSqlProfileProvider"
    automaticSaveEnabled="false" >

     …
  <properties>

     …

     <add name="EmailAddresses"
    type="System.Collections.Specialized.StringCollection"
    serializeAs="Xml"
    allowAnonymous="false"

    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
    </properties>
  </profile>
```

When using collections, you cannot set the `serializeAs` property to `String`. When using the `SqlProfileProvider`, you should set it to `Xml` or `Binary`.

In the code-behind class for our application's demonstration page, we initialize the controls on the form with data from the logged in user's profile. For the simple properties, the `Text` properties for the `Textbox` controls are set with the profile values. For the email address collection, we first add a blank email address to the collection and then bind the data to a `Repeater` control containing `Textbox` controls. Adding the blank email address before binding renders an empty textbox to allow the user to enter an additional email address.

```
Private Sub initializeForm()
  txtFirstName.Text = Profile.FirstName
  txtLastName.Text = Profile.LastName
  txtAddress1.Text = Profile.MailingAddress.Address1
  txtAddress2.Text = Profile.MailingAddress.Address2
  txtCity.Text = Profile.MailingAddress.City
  txtState.Text = Profile.MailingAddress.State
  txtZipCode.Text = Profile.MailingAddress.ZipCode

  'add a blank entry to allow the user to add a new email address
  Profile.EmailAddresses.Add("")

  repEmailAddresses.DataSource = Profile.EmailAddresses
  repEmailAddresses.DataBind()
End Sub 'initializeForm
```

```
  private void initializeForm()
  {
    txtFirstName.Text = Profile.FirstName;
 txtLastName.Text = Profile.LastName;
 txtAddress1.Text = Profile.MailingAddress.Address1;
 txtAddress2.Text = Profile.MailingAddress.Address2;
 txtCity.Text = Profile.MailingAddress.City;
 txtState.Text = Profile.MailingAddress.State;
 txtZipCode.Text = Profile.MailingAddress.ZipCode;

 // add a blank entry to allow the user to add a new email address
 Profile.EmailAddresses.Add("");

 repEmailAddresses.DataSource = Profile.EmailAddresses;
 repEmailAddresses.DataBind();
} // initializeForm
```

When the user clicks the update button, we copy the data from the controls on the form to the user's profile, eliminating any blank email addresses in the process. We then call the `Save` method of the `Profile` object to update the user's profile data in the data store.

The `<profile>` element has an `automaticSaveEnabled` attribute that controls the automatic saving of the profile data. If the `automaticSaveEnabled` attribute is set to `true`, the profile data will be updated at the completion of a page request if any data has been changed. If the `automaticSaveEnabled` attribute is set to `false`, you will be responsible for performing the update.

You should not call the `Save` method of the `Profile` object if the `automaticSaveEnabled` attribute of the `<profile>` element is set to `true`. This can result in a performance impact since the profile data will be updated twice.

As this solution demonstrates, by using Profile features provided with ASP.NET 2.0, you can include user profile data in your application without writing any code to persist the data. In fact, with the provider model used by ASP.NET, your application does not require any knowledge of how the profile data is persisted. With the profile API provided, you can change the profile provider to use a different data store without making any changes to your application code.

See Also

Recipes 9.1, 9.5, 10.2, and 10.3; MSDN Library for information on implementing a custom profile provider

Example 10-1. Modifications to web.config for user profiles

```
<?xml version="1.0"?>
```

```
<configuration>

  <system.web>
    <!--

   The <authentication> section enables configuration
   of the security authentication mode used by
   ASP.NET to identify an incoming user.

   -->
    <authentication mode="Forms" >
     <forms name=".ASPNETCookbookVBPersonalization1"
    loginUrl="Login.aspx"
    protection="All"
     timeout="30"
    path="/"
    defaultUrl="UpdateProfile.aspx" />
  </authentication>

  <authorization>
    <deny users="*" /> <!-- Deny all users -->
  </authorization>

  <membership defaultProvider="AspNetSqlMembershipProvider"
        userIsOnlineTimeWindow="15">
    <providers>
   <remove name="AspNetSqlMembershipProvider" />
   <add name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider"
    connectionStringName="sqlConnectionString"
    enablePasswordRetrieval="false"
    enablePasswordReset="true"
    requiresQuestionAndAnswer="true"
    applicationName="/"
    requiresUniqueEmail="false"
         minRequiredPasswordLength="4"
     minRequiredNonalphanumericCharacters="0"
    passwordFormat="Hashed"
     maxInvalidPasswordAttempts="5"
     passwordAttemptWindow="10"
    passwordStrengthRegularExpression="" />
    </providers>
     </membership>

      <roleManager defaultProvider="AspNetSqlRoleProvider"
          enabled="true"
          cacheRolesInCookie="true"
            cookieName=".ASPNETCookbookVBPersonalization1Roles"
          cookieTimeout="30"
          cookiePath="/"
          cookieRequireSSL="false"
          cookieSlidingExpiration="true"
          cookieProtection="All" >

    <providers>
   <remove name="AspNetSqlRoleProvider" />
   <add name="AspNetSqlRoleProvider"
    type="System.Web.Security.SqlRoleProvider"
    connectionStringName="sqlConnectionString"
    applicationName="/" />
    </providers>
  </roleManager>

  ><profile enabled="true"
    defaultProvider="AspNetSqlProfileProvider"
    automaticSaveEnabled="false" >
    <providers>
      <remove name="AspNetSqlProfileProvider" />
   <add name="AspNetSqlProfileProvider"
    type="System.Web.Profile.SqlProfileProvider"
    connectionStringName="sqlConnectionString"
    applicationName="/" />
    </providers>

    <properties>
      <add name="FirstName"
    type="System.String"
    allowAnonymous="false"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
   <add name="LastName"
    type="System.String"
    serializeAs="String"
    allowAnonymous="false"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
   <group name="MailingAddress">
     <add name="Address1"
    type="System.String"
    serializeAs="String"
    allowAnonymous="false"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
     <add name="Address2"
    type="System.String"
    serializeAs="String"
    allowAnonymous="false"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
     <add name="City"
    type="System.String"
    serializeAs="String"
    allowAnonymous="false"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
     <add name="State"
    type="System.String"
    serializeAs="String"
    allowAnonymous="false"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
     <add name="ZipCode"
    type="System.String"
    serializeAs="String"
    allowAnonymous="false"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
     </group>
     <add name="EmailAddresses"

   type="System.Collections.Specialized.StringCollection"
   serializeAs="Xml"
   allowAnonymous="false"
   provider="AspNetSqlProfileProvider"
   defaultValue=""
   readOnly="false" />
  </properties>
</profile>
  </system.web>
  <!-
  ****************************************************************************
  The following section defines the pages that require authentication
  for access. An entry must be included for each page that requires
  authentication with a list of the roles required for access to
  the page.

  Valid Roles are as follows.
  NOTE: The roles must be entered exactly as listed.

    User
    Admin
  ****************************************************************************
  -->
  <location path="UpdateProfile.aspx">
 <system.web>
  <authorization>
       <allow roles="User,
       Admin"/>
     <deny users="*"/>
  </authorization>
 </system.web>
  </location>

</configuration>
```

Example 10-2. Update user profile data (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="UpdateProfile.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.UpdateProfile"
 Title="User Profile" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  User Profile (VB)
 </div>
 <table width="60%" align="center" border="0">
  <tr>
    <td class="labelNormal">FirstName: </td>
    <td><asp:TextBox ID="txtFirstName" runat="server" /></td>
     </tr>
     <tr>

    <td class="labelNormal">LastName: </td>
    <td><asp:TextBox ID="txtLastName" runat="server" /></td>
     </tr>
     <tr>
        <td class="labelNormal">Address 1: </td>
     <td><asp:TextBox ID="txtAddress1" runat="server" /></td>
     </tr>
     <tr>
        <td class="labelNormal">Address 2: </td>
     <td><asp:TextBox ID="txtAddress2" runat="server" /></td>
     </tr>
     <tr>
       <td class="labelNormal">City: </td>
    <td><asp:TextBox ID="txtCity" runat="server" /></td>
     </tr>
     <tr>
       <td class="labelNormal">State: </td>
    <td><asp:TextBox ID="txtState" runat="server" /></td>
     </tr>
     <tr>
       <td class="labelNormal">Zip Code: </td>
    <td><asp:TextBox ID="txtZipCode" runat="server" /></td>
   </tr>
   <tr>
      <td valign="top" class="labelNormal">Email Addresses: </td>
    <td>
       <table border="0" cellpadding="0" cellspacing="0">
        <asp:Repeater ID="repEmailAddresses" runat="server">
       <ItemTemplate>
         <tr>
        <td>
          <asp:TextBox ID="txtEmailAddress" runat="server"
           Text="<%#Container.DataItem%>" />
        </td>
      </tr>
       </ItemTemplate>
     </asp:Repeater>
      </table>
    </td>
     </tr>
     <tr>
       <td colspan="2" align="center">
      <br />
      <asp:Button ID="btnUpdate" runat="server"
    Text="Update"
    OnClick="btnUpdate_Click" />
      </td>
   </tr>
 </table>
</asp:Content>
```

Example 10-3. Update user profile data (.vb)

```

Option Explicit On
Option Strict On

Imports System
Imports System.Configuration.ConfigurationManager
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
  ''' UpdateProfile.aspx
  ''' </summary>
  Partial Class UpdateProfile
 Inherits System.Web.UI.Page
 '''*********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
  If (Not Page.IsPostBack) Then
      initializeForm( )
     End If
 End Sub 'Page_Load

 '''*********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the update button click
 ''' event. It is responsible for updating the user profile from the data
 ''' on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub btnUpdate_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
  Dim item As RepeaterItem
  Dim txtBox As TextBox
  Dim emailAddress As String
  Dim emailAddresses As StringCollection

  Profile.FirstName = txtFirstName.Text
  Profile.LastName = txtLastName.Text
  Profile.MailingAddress.Address1 = txtAddress1.Text
  Profile.MailingAddress.Address2 = txtAddress2.Text
  Profile.MailingAddress.City = txtCity.Text
  Profile.MailingAddress.State = txtState.Text
  Profile.MailingAddress.ZipCode = txtZipCode.Text

  emailAddresses = New StringCollection
  For Each item In repEmailAddresses.Items
    txtBox = CType(item.FindControl("txtEmailAddress"), _
         TextBox)
    emailAddress = txtBox.Text.Trim
    If (emailAddress.Length > 0) Then
   emailAddresses.Add(emailAddress)
    End If
  Next item

  Profile.EmailAddresses = emailAddresses
  Profile.Save( )

  initializeForm( )
 End Sub 'btnUpdate_Click
```

```
 '''*************************************************************************
 ''' <summary>
 ''' This routine updates the form with data from the user's profile
 ''' </summary>
 Private Sub initializeForm( )
  txtFirstName.Text = Profile.FirstName
  txtLastName.Text = Profile.LastName
  txtAddress1.Text = Profile.MailingAddress.Address1
  txtAddress2.Text = Profile.MailingAddress.Address2
  txtCity.Text = Profile.MailingAddress.City
  txtState.Text = Profile.MailingAddress.State
  txtZipCode.Text = Profile.MailingAddress.ZipCode

  'add a blank entry to allow the user to add a new email address
  Profile.EmailAddresses.Add("")

  repEmailAddresses.DataSource = Profile.EmailAddresses
  repEmailAddresses.DataBind( )
 End Sub 'initializeForm
  End Class 'UpdateProfile
End Namespace
```

Example 10-4. Update user profile data (.cs)

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

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 /// UpdateProfile.aspx
 /// </summary>
 public partial class UpdateProfile : System.Web.UI.Page
 {

   ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
   if (!Page.IsPostBack)
   {
     initializeForm( );
   }
 } // Page_Load

 ///***********************************************************************
 /// <summary>
 /// This routine provides the event handler for the update button click
 /// event. It is responsible for updating the user profile from the data
 /// on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnUpdate_Click(Object sender, System.EventArgs e)
 {
   TextBox txtBox;
   String emailAddress;
   StringCollection emailAddresses;

   Profile.FirstName = txtFirstName.Text;
   Profile.LastName = txtLastName.Text;
   Profile.MailingAddress.Address1 = txtAddress1.Text;
   Profile.MailingAddress.Address2 = txtAddress2.Text;
   Profile.MailingAddress.City = txtCity.Text;
   Profile.MailingAddress.State = txtState.Text;
   Profile.MailingAddress.ZipCode = txtZipCode.Text;

   emailAddresses = new StringCollection( );
   foreach (RepeaterItem item in repEmailAddresses.Items)
   {
     txtBox = (TextBox)(item.FindControl("txtEmailAddress"));
  emailAddress = txtBox.Text.Trim( );
  if (emailAddress.Length > 0)
  {
    emailAddresses.Add(emailAddress);
  }
   }

   Profile.EmailAddresses = emailAddresses;
   Profile.Save( );

   initializeForm( );
 }  // btnUpdate_Click

 ///***********************************************************************
 /// <summary>
 /// This routine updates the form with data from the user's profile
 /// </summary>
 private void initializeForm( )
 {
   txtFirstName.Text = Profile.FirstName;
   txtLastName.Text = Profile.LastName;
   txtAddress1.Text = Profile.MailingAddress.Address1;
   txtAddress2.Text = Profile.MailingAddress.Address2;
   txtCity.Text = Profile.MailingAddress.City;
   txtState.Text = Profile.MailingAddress.State;
   txtZipCode.Text = Profile.MailingAddress.ZipCode;

   // add a blank entry to allow the user to add a new email address
   Profile.EmailAddresses.Add("");

   repEmailAddresses.DataSource = Profile.EmailAddresses;
   repEmailAddresses.DataBind( );
 } // initializeForm
  } // UpdateProfile
}
```

PREV

PREV

Recipe 10.3. Inheriting a Profile

Problem

You want to use ASP.NET 2.0's profile features but you do not want to define the profile information i the *web.config* file because you use the same profile definition in multiple applications.

Solution

Implement the membership features to provide user authentication, as described in Recipe 9.5, create a custom class that inherits from `ProfileBase`, add properties for each user-specific data item you want to include in the profile, and modify the `<profile>` element in *web.config* to set the `inherits` attribute to the custom class you created to store the user's profile data.

Use the .NET language of your choice to create a custom class for the profile data:

1. Inherit from `ProfileBase`.

2. Add a property for each data item to be included in the user profile

Modify *web.config* as follows:

1. Add a `<profile>` element with a `<provider>` element defining the provider used to store and retrieve user-specific data.

2. Set the `inherits` attribute of the `<profile>` element to the fully qualified namespace of the custom class you created.

Example 10-5 shows the modifications made to *web.config* for this solution. The custom user profile class is shown in Examples 10-6 (VB) and 10-7 (C#). The example page that uses the user profile information is identical to the example in Recipe 10.1 and is shown in Examples 10-2, 10-3 through 10-4.

Discussion

If you use the same user profile data in multiple applications, defining the profile in *web.config* creates a maintenance problem when any changes are required since the changes would have to be made in the *web.config* file of each application. Instead, defining the profile in a class that can be compiled into an assembly and then using the assembly in each application is a better solution. The profile infrastructure provided in ASP.NET 2.0 supports using a custom class for profile data to support this situation.

The custom profile class has three specific requirements. First, the class must inherit from `ProfileBase`. Second, the class must be *`serializable`*, meaning that all of the data in the class must be able to be serialized; most .NET data types can be serialized, so this is generally not an issue. Third, the class must include a property for each user profile data item to be stored. The class does not need to provide instance variables to store the class data. Instead, the data for each property must be accessed through the parent class, as shown below.

VB

```
Public Property FirstName( ) As String
  Get
 Return CType(MyBase.Item("FirstName"), String)
  End Get

  Set(ByVal value As String)
 MyBase.Item("FirstName") = value
  End Set
End Property
```

```
public String FirstName
{
  get
  {
 return (String)(base["FirstName"]);
  }
  set
  {
 base["FirstName"] = value;
  }
} // FirstName
```

The base class uses a collection of objects to store the property data using the name of the property. Since the data is stored as an `<Object>`, it must be cast to the appropriate data type when retrieving the data for a property.

The `<profile>` element in the *web.config* file for this solution includes an `inherits` attribute with the value set to the fully qualified namespace of the custom profile class. This provides the definition of the data the `SqlProfileProvider` will persist in the data store.

```
<profile enabled="true"
   defaultProvider="AspNetSqlProfileProvider"
  inherits="ASPNetCookbook.VBExamples.UserProfile"
  automaticSaveEnabled="false" >
  <providers>
<remove name="AspNetSqlProfileProvider" />
<add name="AspNetSqlProfileProvider"
  type="System.Web.Profile.SqlProfileProvider"
  connectionStringName="sqlConnectionString"
  applicationName="/" />
  </providers>
</profile>
```

In the custom profile class for our example, we have emulated the grouping functionality for properties described in Recipe 10.1. We have created a `MailingAddress` property with a data type of `Address`, and we have created an `Address` class with properties for `Address1`, `Address2`, `City`, `State`, and `ZipCode`. By using this approach, additional properties such as `ShippingAddress` can be added without having to define individual properties for each of the elements in an address. In addition, usi this approach, the page used in Recipe 10.1 to demonstrate displaying and updating the profile data can be used with no changes.

See Also

Recipes 9.5 and 10.1

Example 10-5. Modifications to web.config to use an inherited profile

```
<?xml version="1.0"?>
<configuration>

  <system.web>
 <!--
     The <authentication> section enables configuration
     of the security authentication mode used by
     ASP.NET to identify an incoming user.
  -->
 <authentication mode="Forms" >
    <forms name=".ASPNETCookbookVBPersonalization1"
   loginUrl="Login.aspx"

   protection="All"
   timeout="30"
   path="/"
   defaultUrl="UpdateProfile.aspx" />
 </authentication>

 <authorization>
   <deny users="*" /> <!-- Deny all users -->
 </authorization>
```

```
<membership defaultProvider="AspNetSqlMembershipProvider"
    userIsOnlineTimeWindow="15">
  <providers>
    <remove name="AspNetSqlMembershipProvider" />
 <add name="AspNetSqlMembershipProvider"
   type="System.Web.Security.SqlMembershipProvider"
   connectionStringName="sqlConnectionString"
   enablePasswordRetrieval="false"
  enablePasswordReset="true"
  requiresQuestionAndAnswer="true"
  applicationName="/"
  requiresUniqueEmail="false"
   minRequiredPasswordLength="4"
   minRequiredNonalphanumericCharacters="0"
  passwordFormat="Hashed"
   maxInvalidPasswordAttempts="5"
   passwordAttemptWindow="10"
  passwordStrengthRegularExpression="" />
  </providers>
</membership>

<roleManager defaultProvider="AspNetSqlRoleProvider"
    enabled="true"
  cacheRolesInCookie="true"
   cookieName=".ASPNETCookbookVBPersonalization2Roles"
   cookieTimeout="30"
  cookiePath="/"
  cookieRequireSSL="false"
  cookieSlidingExpiration="true"
  cookieProtection="All" >
  <providers>
 <remove name="AspNetSqlRoleProvider" />
 <add name="AspNetSqlRoleProvider"
   type="System.Web.Security.SqlRoleProvider"
   connectionStringName="sqlConnectionString"
  applicationName="/" />
  </providers>
</roleManager>

<profile enabled="true"
    defaultProvider="AspNetSqlProfileProvider"
  inherits="ASPNetCookbook.VBExamples.UserProfile"
  automaticSaveEnabled="false" >
 <providers>
   <remove name="AspNetSqlProfileProvider" />
   <add name="AspNetSqlProfileProvider"
  type="System.Web.Profile.SqlProfileProvider"
  connectionStringName="sqlConnectionString"
  applicationName="/" />
 </providers>
</profile>
 </system.web>
```

```
  <!--
   ****************************************************************************
      The following section defines the pages that require authentication
 for access. An entry must be included for each page that requires
 authentication with a list of the roles required for access to
 the page.

 Valid Roles are as follows.
 NOTE: The roles must be entered exactly as listed.

    User
    Admin
   ****************************************************************************
  -->
  <location path="UpdateProfile.aspx">
 <system.web>
   <authorization>
     <allow roles="User,
         Admin"/>
  <deny users="*"/>
   </authorization>
 </system.web>
  </location>

</configuration>
```

Example 10-6. Custom profile class (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides a container for the the user's profile information
  ''' </summary>
  Public Class UserProfile
 Inherits ProfileBase

 '''*********************************************************************
 ''' <summary>
 ''' This property provides the ability to get/set the firstName
 ''' </summary>
 Public Property FirstName() As String

   Get
 Return CType(MyBase.Item("FirstName"), String)
   End Get
   Set(ByVal value As String)
 MyBase.Item("FirstName") = value
   End Set
 End Property

 '''*************************************************************************
 ''' <summary>
 ''' This property provides the ability to get/set the lastName
 ''' </summary>
 Public Property LastName() As String
   Get
  Return CType(MyBase.Item("LastName"), String)
   End Get

   Set(ByVal value As String)
  MyBase.Item("LastName") = value
   End Set
 End Property

 '''*************************************************************************
 ''' <summary>
 ''' This property provides the ability to get/set the mailing address
 ''' </summary>
 Public Property MailingAddress() As Address
   Get
  Return CType(MyBase.Item("MailingAddress"), Address)
   End Get

   Set(ByVal value As Address)
  MyBase.Item("MailingAddress") = value
   End Set
 End Property

 '''*************************************************************************
 ''' <summary>
 ''' This property provides the ability to get/set the email addresses
 ''' </summary>
 Public Property EmailAddresses() As StringCollection
   Get
  Return CType(MyBase.Item("EmailAddresses"), StringCollection)
   End Get

   Set(ByVal value As StringCollection)
  MyBase.Item("EmailAddresses") = value
   End Set
 End Property
End Class 'UserProfile

 ''' <summary>
 ''' The following class is used as a data container for address data
 ''' </summary>
 Public Class Address
  Private mAddress1 As String
  Private mAddress2 As String
  Private mCity As String
  Private mState As String
  Private mZipCode As String

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the address1
  ''' </summary>
  Public Property Address1() As String
   Get
    Return (mAddress1)
   End Get

   Set(ByVal value As String)
    mAddress1 = value
   End Set
  End Property

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the address2
  ''' </summary>
  Public Property Address2() As String
   Get
    Return (mAddress2)
   End Get

   Set(ByVal value As String)
    mAddress2 = value
   End Set
  End Property

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the city
  ''' </summary>
  Public Property City() As String
   Get
    Return (mCity)
   End Get

   Set(ByVal value As String)
    mCity = value
   End Set
 End Property

   '''*************************************************************************
 ''' <summary>
 ''' This property provides the ability to get/set the state
 ''' </summary>
```

```
 Public Property State() As String
   Get
  Return (mState)
   End Get

   Set(ByVal value As String)
  mState = value
   End Set
 End Property

 '''**********************************************************************
 ''' <summary>
 ''' This property provides the ability to get/set the state
 ''' </summary>
 Public Property ZipCode() As String
   Get
  Return (mZipCode)
   End Get

   Set(ByVal value As String)
  mZipCode = value
   End Set
 End Property
  End Class 'Address
End Namespace
```

Example 10-7. Custom profile class (.cs)

```
using System;
using System.Collections.Specialized;
using System.Web.Profile;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides a container for the the user's profile information.
 /// </summary>
 public class UserProfile : ProfileBase
 {
   ///**********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the firstName
  /// </summary>
  public String FirstName
  {
   get
   {
      return (String)(base["FirstName"]);

    }
    set
    {
      base["FirstName"] = value;
    }
  } // FirstName

  ///*************************************************************************
  /// <summary>
  /// This property provides the ability to get/set the lastName
  /// </summary>
  public String LastName
  {
    get
    {
      return (String)(base["LastName"]);
    }
    set
    {
      base["LastName"] = value;
    }
  } // LastName

  ///*************************************************************************
  /// <summary>
  /// This property provides the ability to get/set the mailing address
  /// </summary>
  public Address MailingAddress
  {
    get
    {
      return (Address)(base["MailingAddress"]);
    }
    set
    {
      base["MailingAddress"] = value;
    }
  } // MailingAddress

  ///*************************************************************************
  /// <summary>
  /// This property provides the ability to get/set the email addresses
  /// </summary>
  public StringCollection EmailAddresses
  {
    get
    {
      return (StringCollection)(base["EmailAddresses"]);
    }
    set
```

```
   {
     base["EmailAddresses"] = value;

   }
  } // EmailAddresses
 } // UserProfile

 /// <summary>
 /// The following class is used as a data container for address data
 /// </summary>
 public class Address
 {
  private String mAddress1;
  private String mAddress2;
  private String mCity;
  private String mState;
  private String mZipCode;

  ///*********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the address1
  /// </summary>
  public String Address1
  {
   get
   {
     return (mAddress1);
   }
   set
   {
     mAddress1 = value;
   }
  } // Address1

  ///*********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the address2
  /// </summary>
  public String Address2
  {
   get
   {
     return (mAddress2);
   }
   set
   {
     mAddress2 = value;
   }
  } // Address2

  ///*********************************************************************
  /// <summary>
```

```
    /// This property provides the ability to get/set the city
    /// </summary>
    public String City

    {
     get
     {
       return (mCity);
     }
     set
     {
       mCity = value;
     }
    } // City

    ///*********************************************************************
    /// <summary>
    /// This property provides the ability to get/set the state
    /// </summary>
    public String State
    {
     get
     {
       return (mState);
     }
     set
     {
       mState = value;
     }
    } // State

    ///*********************************************************************
    /// <summary>
    /// This property provides the ability to get/set the state
    /// </summary>
    public String ZipCode
    {
     get
     {
       return (mZipCode);
     }
     set
     {
       mZipCode = value;
     }
    } // ZipCode
   } // Address
  }
```

PREV

Recipe 10.4. Using and Migrating Anonymous Profiles

Problem

You want to store profile information for users who are not logged into your application.

Solution

Implement the solution described in Recipe 10.1, add an `<anonymousIdentification>` element to the *web.config* file to enable anonymous identification, and set the `allow-Anonymous` attributes of the profile property elements to true in *web.config* for each of the profile properties that should be stored for anonymous users.

Modify *web.config* as follows:

1. Add an `<anonymousIdentification>` element to enable the anonymous identification features and to define the cookie used to identify the user.
2. Set the `allowAnonymous` attribute of each profile property that is to be persisted for anonymous users to `true`.

Example 10-8 shows the modifications made to *web.config* for this solution. The example page that uses the user profile information is identical to the example in Recipe 10.1 and is shown in Examples 10-2, 10-3 through 10-4.

Discussion

Storing anonymous user profile data was possible in ASP.NET 1.x, but, since there was no inherent support in 1.x, you had to implement it yourself. As part of the profile features, ASP.NET 2.0 provide full support for storing data for anonymous users. Like storing profile data for authenticated users, it can be configured with little code.

When anonymous identification is enabled, ASP.NET 2.0 generates a GUID to use as the identity for visitors to your application on their first visit. This GUID is stored in a cookie that is persisted on the user's computer. For all subsequent page requests and visits to your application, the cookie is retrieved and the GUID is used to access his or her profile information. Other than relying on a cooki to identify the user (instead of logging into your application), the mechanism for managing the user profile data is the same whether the user is anonymous or authenticated.

ASP.NET supports using the URL for the anonymous identification data; however, unless the user bookmarks the URL with the encoded data, the user cannot be identified on subsequent visits to your application.

Since cookies are not shared between browsers, users who use multiple browsers may have unexpected results when accessing an application with one browser and then another. If your application uses anonymous profiles and a user first accesses it with Internet Explorer, for example, and then accesses with Firefox, two separate profiles will be created and stored for the user. If the user changes the profile information using one browser, those changes will not be reflected when using the other browser.

To enable anonymous identification, you need to add an `<anonymousIdentification>` element to *web.config*. Multiple attributes are provided to configure anonymous identification, as shown in Table 10-3.

Table 10-3. anonymousIdentification element attributes

Attribute	Description
`enabled`	Specifies if the `HttpAnonymousIdentificationModule` is enabled. Valid values are `true` and `false`. The default value is `false`.
`cookieName`	Defines the name of the cookie used to store the anonymous ID of the user. The default value is `.ASPXANONYMOUS`.
`cookiePath`	Defines the path to the directory where the cookie is stored. Most browsers treat the path as case-sensitive and will not return the cookie for a request that does not match the value provided for the path attribute. Unless your application requires specifying the path, leave the path as "/".
`cookieRequireSSL`	Defines whether an SSL connection is required to transmit the cookie to the client. Valid values are `true` and `false`. The default value is `false`.
`cookieTimeout`	Specifies the cookie expiration time in minutes.
`cookieSlidingExpiration`	Defines if sliding expiration of the cookie is enabled. If set to `TRue`, the cookie time to expire will be refreshed on every page request. If set to `false`, the cookie will expire at the time set when the cookie was created. The default value is `TRue`.
`cookieProtection`	Defines the protection method used for the cookie. Possible values are `All, None, Encryption`, and `Validation. Validation` specifies that the cookie data will be validated to ensure it was not altered in transit. `Encryption` specifies that the cookie is encrypted. `All` specifies that data validation and encryption will be used. `None` specifies that no protection

Attribute	Description
	will be provided for the cookie information. The default is `All` and is recommended because it offers the highest level of protection for the cookie.

In some applications, supporting conversion of an anonymous user to an authenticated user is necessary. For example, when a user fills his shopping cart and then logs in to make a purchase, you want to transfer the data from the shopping cart in the anonymous profile to the one in the authenticated profile.

ASP.NET 2.0 provides the ability for you to perform the conversion by firing an event when an anonymous user logs into your application. The event is the `OnMigrateAnonymous` and must be handled in *Global.asax*, as shown in Example 10-9 (VB) and Example 10-10 (C#).

The first step to migrate an anonymous profile to an authenticated user profile is to get the user's anonymous profile data. You can do this by calling the `GetProfile` method of the `Profile` class passing the ID of the anonymous user. The anonymous user ID is passed to the `Profile_OnMigrateAnonymous` method in the `args` parameter.

```
anonymousProfile = Profile.GetProfile(args.AnonymousID)

anonymousProfile = Profile.GetProfile(args.AnonymousID);
```

Next, you need to verify that the authenticated user profile does not exist. Since the profile for a user, whether anonymous or authenticated, will never be `null (Nothing` in VB), you can check the `LastActivityDate` property to determine if it has a valid date. If the date is equal to the minimum date value, the authenticated profile does not exist. Any other value indicates an authenticated profile exists.

```
If (Profile.LastActivityDate = DateTime.MinValue) Then

  …

End If


if (Profile.LastActivityDate == DateTime.MinValue)
{
  …
}
```

If you do not check the authenticated user profile in the `OnMigrateAnonymous` event handler before migrating the anonymous user profile, the authenticated user profile will be overwritten every time the user logs in, since the `OnMigrateAnonymous` event is fired every time the user logs in to your application.

After determining that the authenticated profile does not exist, you need to copy the desired data from the anonymous profile to the authenticated profile and save the profile.

VB

```
If (Profile.LastActivityDate = DateTime.MinValue) Then
  'logged in user's profile does not exist so copy the anonymous
  'profile data to the logged in user's profile
  Profile.FirstName = anonymousProfile.FirstName
  Profile.LastName = anonymousProfile.LastName
   Profile.MailingAddress.Address1  =  anonymousProfile.MailingAddress.Address1
   Profile.MailingAddress.Address2  =  anonymousProfile.MailingAddress.Address2
  Profile.MailingAddress.City = anonymousProfile.MailingAddress.City
  Profile.MailingAddress.State = anonymousProfile.MailingAddress.State
  Profile.MailingAddress.ZipCode = anonymousProfile.MailingAddress.ZipCode
  Profile.EmailAddresses = anonymousProfile.EmailAddresses

  'save the logged in user's profile
  Profile.Save()
End If
```

```
if (Profile.LastActivityDate == DateTime.MinValue)
{
  // logged in user's profile does not exist so copy the anonymous
  // profile data to the logged in user's profile
  Profile.FirstName = anonymousProfile.FirstName;
  Profile.LastName = anonymousProfile.LastName;
   Profile.MailingAddress.Address1  =  anonymousProfile.MailingAddress.Address1;
   Profile.MailingAddress.Address2  =  anonymousProfile.MailingAddress.Address2;

  Profile.MailingAddress.City = anonymousProfile.MailingAddress.City;
  Profile.MailingAddress.State = anonymousProfile.MailingAddress.State;
  Profile.MailingAddress.ZipCode = anonymousProfile.MailingAddress.ZipCode;
  Profile.EmailAddresses = anonymousProfile.EmailAddresses;

  // save the logged in user's profile
  Profile.Save();
}
```

Next, you should delete the anonymous profile to help in managing unneeded profiles. This is important since profile data is permanent and is never automatically purged (see Recipe 10.4 for an example of managing profiles).

The final step is to delete the cookie used to identify the anonymous user since it is unneeded.

> If you do not delete the anonymous cookie, the `OnMigrateAnonymous` event will fire on every page access. This is caused by the way ASP.NET determines if the event should be fired. If the user is authenticated and the anonymous cookie exists, ASP.NET will assume this is the initial login and fire the event. If the cookie is never deleted, the event will fire every time a page is requested.

The infrastructure provided by ASP.NET 2.0 for profiles will meet the needs of most applications with little or no coding required. This can be a significant savings in developing your application.

See Also

Recipes 10.1 and 10.4

Example 10-8. Modifications to web.config to support profiles for anonymous users

```
<?xml version="1.0"?>
<configuration>

<system.web>
 <!--
   The <authentication> section enables configuration
   of the security authentication mode used by
   ASP.NET to identify an incoming user.
   -->
 <authentication mode="Forms" >
    <forms name=".ASPNETCookbookVBPersonalization3"
   loginUrl="Login.aspx"
   protection="All"
    timeout="30"
   path="/"
   defaultUrl="UpdateProfile.aspx" />
 </authentication>

 <authorization>
   <allow users="*" /> <!-- Allow all users -->
 </authorization>

 <membership defaultProvider="AspNetSqlMembershipProvider"
     userIsOnlineTimeWindow="15">
   <providers>
     <remove name="AspNetSqlMembershipProvider" />
```

```
  <add name="AspNetSqlMembershipProvider"
    type="System.Web.Security.SqlMembershipProvider"
    connectionStringName="sqlConnectionString"
    enablePasswordRetrieval="false"
    enablePasswordReset="true"
    requiresQuestionAndAnswer="true"
    applicationName="/AnonymousVB"
    requiresUniqueEmail="false"
     minRequiredPasswordLength="4"
     minRequiredNonalphanumericCharacters="0"
    passwordFormat="Hashed"
     maxInvalidPasswordAttempts="5"
     passwordAttemptWindow="10"
    passwordStrengthRegularExpression="" />
   </providers>
 </membership>

 <roleManager defaultProvider="AspNetSqlRoleProvider"
    enabled="true"
    cacheRolesInCookie="true"
     cookieName=".ASPNETCookbookVBPersonalization1Roles"
     cookieTimeout="30"
    cookiePath="/"
    cookieRequireSSL="false"
    cookieSlidingExpiration="true"
    cookieProtection="All" >
 <providers>
   <remove name="AspNetSqlRoleProvider" />
   <add name="AspNetSqlRoleProvider"
    type="System.Web.Security.SqlRoleProvider"
    connectionStringName="sqlConnectionString"
    applicationName="/AnonymousVB" />
 </providers>
</roleManager>

<anonymousIdentification enabled="true"
    cookieName="ASPNETCookbookVBAnonymousID"
    cookiePath="/"
    cookieRequireSSL="false"
    cookieTimeout="525600"
    cookieSlidingExpiration="true"
    cookieProtection="All" />

 <profile enabled="true"
    defaultProvider="AspNetSqlProfileProvider"
    automaticSaveEnabled="false" >
   <providers>
  <remove name="AspNetSqlProfileProvider" />
  <add name="AspNetSqlProfileProvider"
    type="System.Web.Profile.SqlProfileProvider"
    connectionStringName="sqlConnectionString"
    applicationName="/AnonymousVB" />
   </providers>

   <properties>
  <add name="FirstName"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
  <add name="LastName"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
  <add name="Theme"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
   <group name="MailingAddress">
  <add  name="Address1"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
  <add  name="Address2"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
  <add name="City"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"

    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
  <add name="State"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
  <add name="ZipCode"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
  </group>
  <add name="EmailAddresses"
   type="System.Collections.Specialized.StringCollection"
   serializeAs="Xml"
   allowAnonymous="true"
   provider="AspNetSqlProfileProvider"
   defaultValue=""
   readOnly="false" />
   </properties>
 </profile>

</system.web>
</configuration>
```

Example 10-9. OnMigrateAnonymous event handler in Global.asax (.vb)

```

Public Sub Profile_OnMigrateAnonymous(ByVal sender As Object, _
        ByVal args As ProfileMigrateEventArgs)
  Dim anonymousProfile As ProfileCommon

  'get the anonymous user's profile
  anonymousProfile = Profile.GetProfile(args.AnonymousID)

  'check to see if the logged in user's profile is already active
   If (Profile.LastActivityDate = DateTime.MinValue) Then
  'logged in user's profile does not exist so copy the anonymous
  'profile data to the logged in user's profile
  Profile.FirstName = anonymousProfile.FirstName
  Profile.LastName = anonymousProfile.LastName
   Profile.MailingAddress.Address1  =  anonymousProfile.MailingAddress.Address1
   Profile.MailingAddress.Address2  =  anonymousProfile.MailingAddress.Address2
  Profile.MailingAddress.City = anonymousProfile.MailingAddress.City
  Profile.MailingAddress.State = anonymousProfile.MailingAddress.State
  Profile.MailingAddress.ZipCode = anonymousProfile.MailingAddress.ZipCode
  Profile.EmailAddresses = anonymousProfile.EmailAddresses

  'save the logged in user's profile
  Profile.Save()
```

```
 End If

 'delete the anonymous user data from the database
 ProfileManager.DeleteProfile(anonymousProfile.UserName)

 'delete the anonymous cookie so this event will no longer
 'fire for a logged-in user
 AnonymousIdentificationModule.ClearAnonymousIdentifier()
End Sub 'Profile_OnMigrateAnonymous
```

Example 10-10. OnMigrateAnonymous event handler in Global.asax (C#)

```
public void Profile_OnMigrateAnonymous(Object sender,
         ProfileMigrateEventArgs args)
{
 ProfileCommon anonymousProfile;

 // get the anonymous user's profile
 anonymousProfile = Profile.GetProfile(args.AnonymousID);

 // check to see if the logged in user's profile is already active
 if (Profile.LastActivityDate == DateTime.MinValue)
 {
  // logged in user's profile does not exist so copy the anonymous
  // profile data to the logged in user's profile
  Profile.FirstName = anonymousProfile.FirstName;
  Profile.LastName = anonymousProfile.LastName;
   Profile.MailingAddress.Address1  =  anonymousProfile.MailingAddress.Address1;
   Profile.MailingAddress.Address2  =  anonymousProfile.MailingAddress.Address2;
  Profile.MailingAddress.City = anonymousProfile.MailingAddress.City;
  Profile.MailingAddress.State = anonymousProfile.MailingAddress.State;
  Profile.MailingAddress.ZipCode = anonymousProfile.MailingAddress.ZipCode;
  Profile.EmailAddresses = anonymousProfile.EmailAddresses;

  // save the logged in user's profile
  Profile.Save();
 }

 // delete the anonymous user data from the database
 ProfileManager.DeleteProfile(anonymousProfile.UserName);

 // delete the anonymous cookie so this event will no longer
 // fire for a logged-in user
 AnonymousIdentificationModule.ClearAnonymousIdentifier();
} // Profile_OnMigrateAnonymous
```

PREV

Recipe 10.5. Managing User Profiles

Problem

You want to provide a mechanism to periodically remove user profiledata that is no longer being used.

Solution

Use the `ProfileManager` class to find inactive profiles, display them in an ASP.NET page for the manager to review, and then use the `ProfileManager` to delete the desired profiles.

In the *.aspx* file:

1. Add a drop-down list or other control to allow selection of the authentication option.
2. Add a drop-down list or other control to allow selection of theinactive period.
3. Add a button to initiate the search for the profile.
4. Add a `GridView` to display the profiles matching the search parameters.
5. Provide the ability to delete one or more profiles.

In the code-behind class for the page, use the .NET language of your choice as follows:

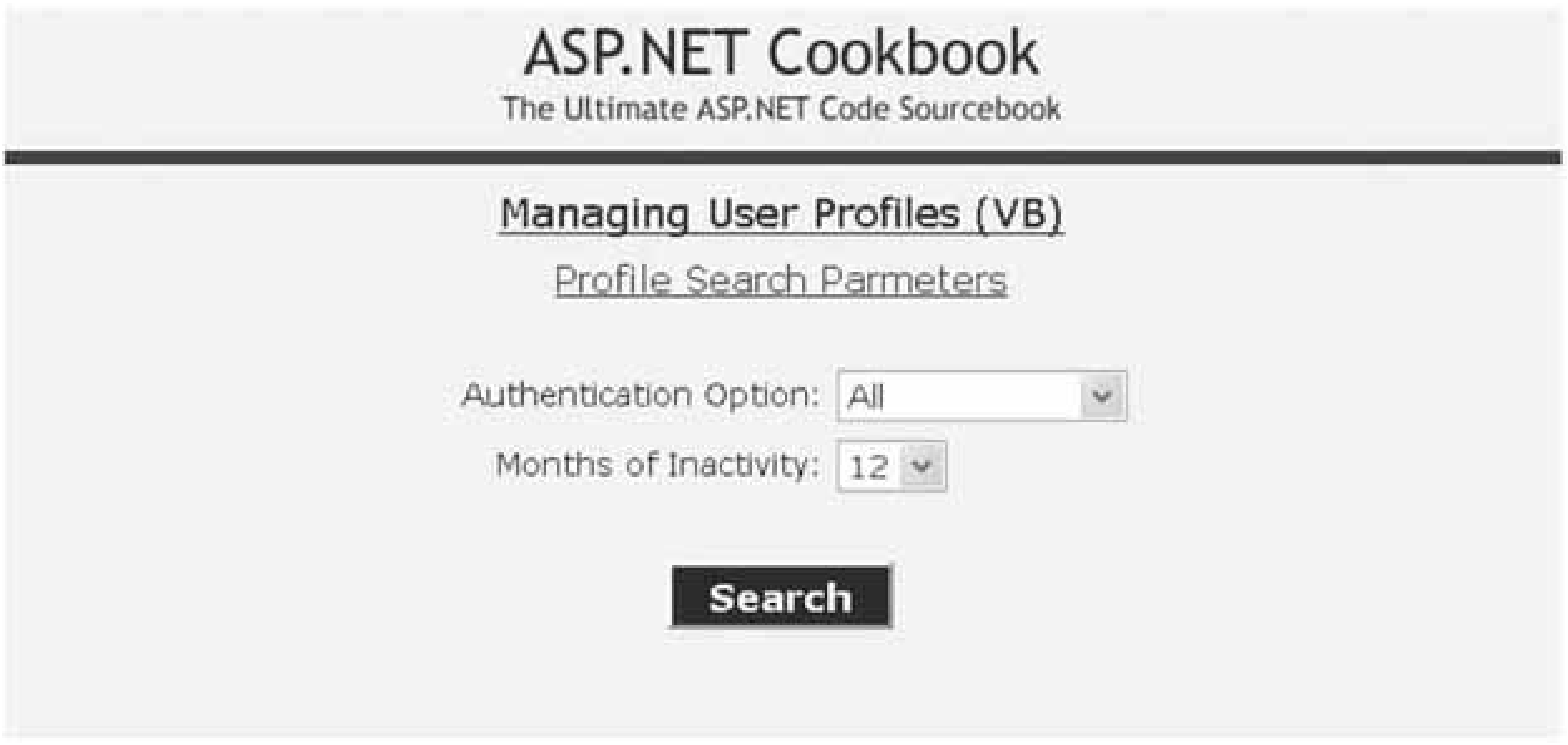
1. Initialize the authentication option drop-down control with the available options.
2. Initialize the inactive period drop-down control with the available options.
3. In the event handler for the Search button click event, use the`GetAllInactive-Profiles` method of the `ProfileManager` class to get the profiles matching the search parameters and then bind the profile data to a `GridView` .
4. In the event handler for the Delete All button click event, use the`DeleteInactiveProfiles` method of the `ProfileManager` class to delete the displayed profiles.

The code we have created to illustrate this solution is shown in Examples 10-11, 10-12 through 10-13 . Example 10-11 shows the *.aspx* file for the page used to manage profiles. The code-behind classes for the page are shown in Examples 10-12(VB) and 10-13 (C#). The output of the page used to manage profiles is shown in Figures 10-1(before searching) and 10-2 (after searching).

Discussion

The Profile provider supplied by Microsoft with ASP.NET 2.0 permanently stores the profile data for anonymous and authenticated user profiles. Depending on the number of users of your application and the number of those who are recurring users, a large number of inactive profiles may be stored in your database. To keep the performance of your application at its peak, you should periodically remove unused profiles.

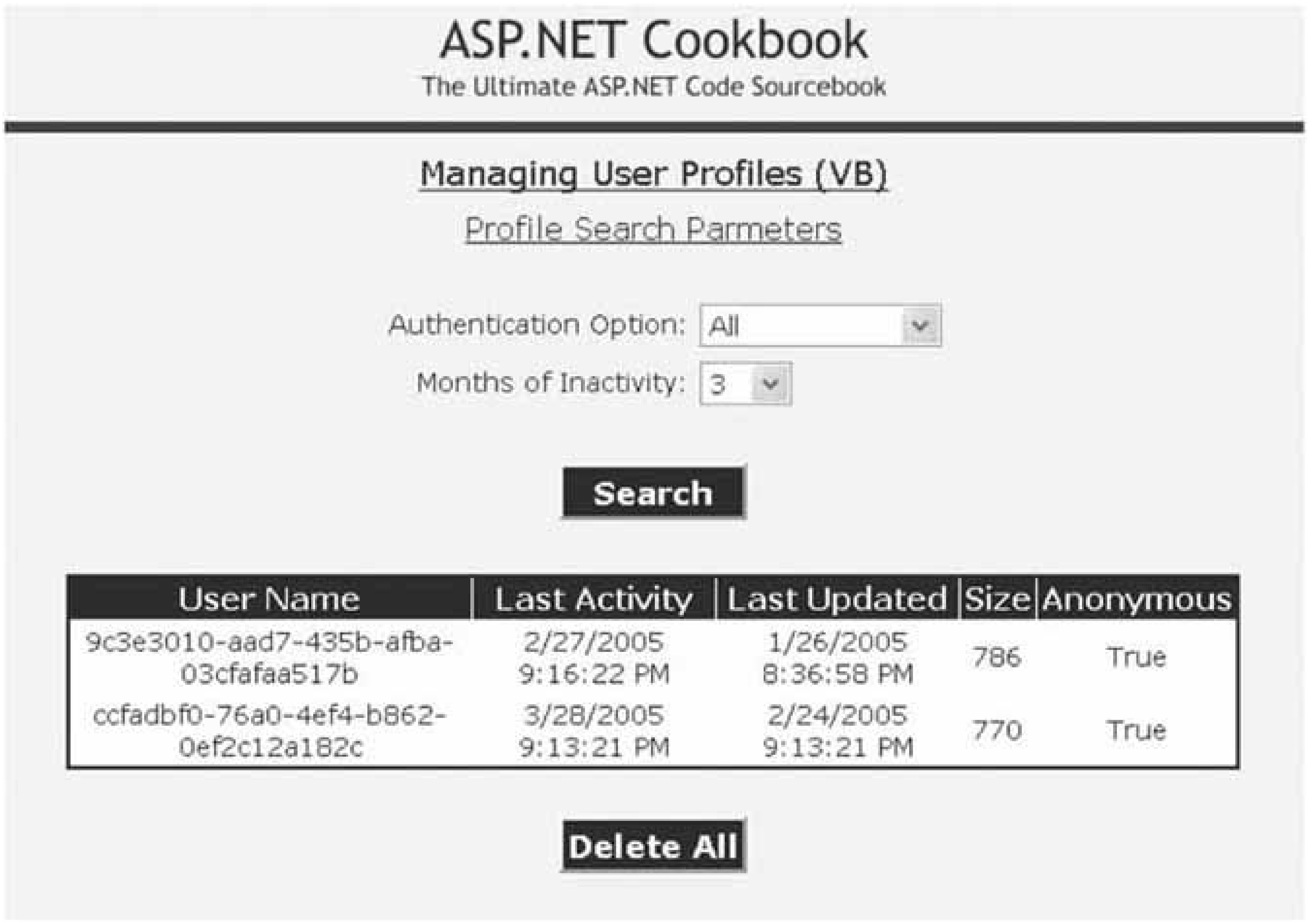
Figure 10-1. Managing profiles page before performing search


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Managing User Profiles (VB)
Profile Search Parmeters
Search

Authentication Option: All

Months of Inactivity: 12

Figure 10-2. Managing profiles page after performing search


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Managing User Profiles (VB)

Profile Search Parmeters

Authentication Option: All

Months of Inactivity: 3

User Name	Last Activity	Last Updated	Size	Anonymous
9c3e3010-aad7-435b-afba-03cfafaa517b	2/27/2005 9:16:22 PM	1/26/2005 8:36:58 PM	786	True
ccfadbf0-76a0-4ef4-b862-0ef2c12a182c	3/28/2005 9:13:21 PM	2/24/2005 9:13:21 PM	770	True

ASP.NET 2.0 provides the ability to manage profiles with the `ProfileManager` class. This class gives you the ability to find profiles matching selected criteria and to delete them using the same criteria o by culling them from a list of users.

In the example we have written to illustrate this solution, we have created a page that provides the ability for the user to enter the authentication type and the number of inactive months. When the user clicks the Search button, the last activity date is calculated from the number of inactive months selected in the Search button event handler. The `GetAllInactiveProfiles` method of the `ProfileManager` class is then called, passing the authentication type and the last activity date.

The `GetAllInactiveProfiles` method returns a `ProfileInfoCollection` that can be bound to a `GridView` or other grid control as shown in Examples 10-12(VB) and 10-13 (C#).

> The `ProfileManager` class has an overloaded `GetAllInactiveProfiles` method that fully supports paginating data. It provides the ability to pass the page size and desired page number to allow retrieving a single page of profiles. In addition, it has a `TotalRecords` parameter that returns the total number of profiles meeting the search criteria.

We store the search parameters in the `ViewState` to use when the user clicks the Delete All button. By storing the parameters used to display the data, we ensure the deleted profiles are the ones being displayed. If we used the selections from the dropdowns on the form, the user may have changed th selection, which would result in a different set of profiles being deleted.

When the user clicks the Delete All button, the search parameters stored in the `ViewState` are retrieved from the `ViewState` in the Delete All button event handler. The `DeleteInactiveProfiles` method of the `ProfileManager` class is then called, passing the search parameters retrieved from the `ViewState` .

The `DeleteInactiveProfiles` method returns the number of profiles deleted. Next, we hide the `GridView` and display a message indicating the number of profiles deleted.

> Like the other delete profile methods of the `ProfileManager` class, the `DeleteInactiveProfile` method deletes only the user profile data. It does not delete the user data from the database. If you want to delete the user data in addition to the profile data, you will need to use the `DeleteUser` method of the `Membership` class.

See Also

Recipe 10.1

Example 10-11. Managing profiles (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH10ManagingUserProfilesVB.aspx.vb"

 Inherits="ASPNetCookbook.VBExamples.CH10ManagingUserProfilesVB"
 Theme="Blue"
 Title="Managing User Profiles" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Managing User Profiles (VB)
 </div>
 <div align="center" class="subHeading">
  Profile Search Parmeters
 </div>
 <br />
 <table width="60%" align="center" border="0" cellpadding="2">
  <tr>
   <td class="labelText" align="right">Authentication Option: </td>
   <td>
    <asp:DropDownList ID="ddAuthenticationOptions" runat="server"
       CssClass="labelText" />
   </td>
  </tr>
  <tr>
   <td class="labelText" align="right">Months of Inactivity: </td>
```

```
    <td>
     <asp:DropDownList ID="ddInactiveMonths" runat="server"
        CssClass="labelText" />

    </td>
   </tr>
   <tr>
    <td colspan="2" align="center">
     <br />
     <asp:Button ID="btnSearch" runat="server"
        Text="Search"
        OnClick="btnSearch_Click" />
    </td>
   </tr>
  </table>

  <div id="divProfiles" runat="server" align="center">
   <br />
   <asp:GridView ID="gvProfiles" runat="server"
       BorderColor="#000080"
       BorderWidth="2px"
       AutoGenerateColumns="False"
       HorizontalAlign="center"
       Width="90%"
       CellPadding="2" >
    <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
    <RowStyle cssClass="tableCellNormal" />
    <AlternatingRowStyle cssClass="tableCellAlternating" />
    <Columns>
     <asp:BoundField DataField="UserName" HeaderText="User Name" />
     <asp:BoundField DataField="LastActivityDate" HeaderText="Last Activity" />
     <asp:BoundField DataField="LastUpdatedDate" HeaderText="Last Updated" />

     <asp:BoundField DataField="Size" HeaderText="Size" />
     <asp:BoundField DataField="IsAnonymous" HeaderText="Anonymous" />
    </Columns>
   </asp:GridView>
   <br />
   <asp:Button ID="btnDeleteAll" runat="server"
        Text="Delete All"
        OnClick="btnDeleteAll_Click" />
  </div>

  <div id="divMessage" runat="server" align="center" class="labelText">
   <br />
   <asp:Literal ID="litMessage" runat="server" />
  </div>
 </asp:Content>
```

Example 10-12. Managing profiles code-behind (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH10ManagingUserProfilesVB.aspx
 ''' </summary>
 Partial  Class  CH10ManagingUserProfilesVB
   Inherits System.Web.UI.Page

   'the following constants define the name of variables in the ViewState
   'used to store the search criteria for the currently displayed
   'profile data
   Private Const VS_AUTHENTICATION_OPTION As String = "AuthenticationOption"
   Private Const VS_LAST_ACTIVITY_DATE As String = "lastActivityDate"

   '''*************************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the page load event. It
   ''' is responsible for initializing the controls on the page.
   ''' </summary>
   '''
   ''' <param name="sender">Set to the sender of the event</param>
   ''' <param name="e">Set to the event arguments</param>
   Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
  Dim item As ListItem

  If (Not Page.IsPostBack) Then
   'add items to the authentication option dropdown
   ddAuthenticationOptions.Items.Clear()
   item = New ListItem("All", _
      (CInt(ProfileAuthenticationOption.All)).ToString())

   ddAuthenticationOptions.Items.Add(item)
   item = New ListItem("Anonymous", _
    (CInt(ProfileAuthenticationOption.Anonymous)).ToString())
   ddAuthenticationOptions.Items.Add(item)
   item = New ListItem("Authenticated", _
    (CInt(ProfileAuthenticationOption.Authenticated)).ToString())
   ddAuthenticationOptions.Items.Add(item)

   'add itesm to the "months of inactivity" dropdown
   ddInactiveMonths.Items.Clear()
   item = New ListItem("0", "0")
   ddInactiveMonths.Items.Add(item)
   item = New ListItem("3", "3")
```

```
    ddInactiveMonths.Items.Add(item)
    item = New ListItem("6", "6")
    ddInactiveMonths.Items.Add(item)
    item = New ListItem("12", "12")
    ddInactiveMonths.Items.Add(item)

    'select the "12 months" entry
    ddInactiveMonths.SelectedIndex = ddInactiveMonths.Items.Count - 1

    'initially hide the grid and message panels
    divProfiles.Visible = False
    divMessage.Visible = False
  End If
End Sub 'Page_Load

'''*************************************************************************
''' <summary>
''' This routine provides the event handler for the search button click
''' event. It is responsible for searching for and displaying the
''' profiles matching the selected criteria
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnSearch_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
  Dim inactiveMonths As Integer
  Dim lastActivityDate As DateTime = DateTime.MinValue
  Dim profileData As ProfileInfoCollection
  Dim authenticationOption As ProfileAuthenticationOption

  authenticationOption = CType(ddAuthenticationOptions.SelectedValue, _
       ProfileAuthenticationOption)
  'calculate the last activity date
  inactiveMonths = CInt(ddInactiveMonths.SelectedValue)
  lastActivityDate = DateTime.Now.AddMonths(-inactiveMonths)

  'get profiles matching authentication option and last activity date
profileData = _
    ProfileManager.GetAllInactiveProfiles(authenticationOption, _
          lastActivityDate)

'check to see if any profiles match the selected criteria
If (profileData.Count = 0) Then
    divProfiles.Visible = False
    divMessage.Visible = True
    litMessage.Text = "No profiles were found matching the entered criteria"
Else
    divProfiles.Visible = True
    divMessage.Visible = False
    gvProfiles.DataSource = profileData
    gvProfiles.DataBind()
```

```
 End If

 'store the search criteria in the ViewState to support deleting
 'profiles when required
 'NOTE: The data is stored to ensure it matches the displayed
 '    profiles to handle the case where the user changes the
 '    criteria and then clicks the delete all button
 ViewState(VS_AUTHENTICATION_OPTION) = authenticationOption
 ViewState(VS_LAST_ACTIVITY_DATE) = lastActivityDate
End Sub 'btnSearch_Click

'''***************************************************************************
''' <summary>
''' This routine provides the event handler for the delete all button
''' click event. It is responsible for deleting the displayed profiles
''' </summary>
 '''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnDeleteAll_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
 Dim lastActivityDate As DateTime
 Dim authenticationOption As ProfileAuthenticationOption
 Dim profilesDeleted As Integer

 'get the search criteria used to display the current data
 lastActivityDate = CDate(ViewState(VS_LAST_ACTIVITY_DATE))
 authenticationOption = CType(ViewState(VS_AUTHENTICATION_OPTION), _
      ProfileAuthenticationOption)

 'delete the profiles
 profilesDeleted = _
   ProfileManager.DeleteInactiveProfiles(authenticationOption, _
        lastActivityDate)

 'hide the grid and output message indicating the number of
 'profiles deleted

  divProfiles.Visible = False
  divMessage.Visible = True
  litMessage.Text = profilesDeleted.ToString() & _
      " Profile(s) were deleted"
 End Sub 'btnDeleteAll_Click
  End Class 'CH10ManagingUserProfilesVB
End Namespace
```

Example 10-13. Managing profiles code-behind (.cs)

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

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH10ManagingUserProfilesCS.aspx
 /// </summary>
 public partial class CH10ManagingUserProfilesCS : System.Web.UI.Page
 {

   // the following constants define the name of variables in the ViewState
   // used to store the search criteria for the currently displayed
   // profile data
   private const String VS_AUTHENTICATION_OPTION = "AuthenticationOption";
   private const String VS_LAST_ACTIVITY_DATE = "lastActivityDate";

    ///************************************************************************
   /// <summary>
   /// This routine provides the event handler for the page load event.
   /// It is responsible for initializing the controls on the page.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void Page_Load(object sender, EventArgs e)
   {
     ListItem item;

  if (!Page.IsPostBack)
  {
    // add items to the authentication option dropdown
    ddAuthenticationOptions.Items.Clear();
    item = new ListItem("All",
    ((int)ProfileAuthenticationOption.All).ToString());
    ddAuthenticationOptions.Items.Add(item);
    item = new ListItem("Anonymous",
    ((int)ProfileAuthenticationOption.Anonymous).ToString());
    ddAuthenticationOptions.Items.Add(item);

    item = new ListItem("Authenticated",
    ((int)ProfileAuthenticationOption.Authenticated).ToString());
    ddAuthenticationOptions.Items.Add(item);

    // add itesm to the "months of inactivity" dropdown
    ddInactiveMonths.Items.Clear();
    item = new ListItem("0", "0");
    ddInactiveMonths.Items.Add(item);
    item = new ListItem("3", "3");
    ddInactiveMonths.Items.Add(item);
```

```
     item = new ListItem("6", "6");
     ddInactiveMonths.Items.Add(item);
     item = new ListItem("12", "12");
     ddInactiveMonths.Items.Add(item);

     // select the "12 months" entry
     ddInactiveMonths.SelectedIndex = ddInactiveMonths.Items.Count - 1;

     // initially hide the grid and message panels
     divProfiles.Visible = false;
     divMessage.Visible = false;

   }
 } // Page_Load

 ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the search button click
 /// event. It is responsible for searching for and displaying the
 /// profiles matching the selected criteria
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnSearch_Click(Object sender,
          System.EventArgs e)
 {
   int inactiveMonths;
   DateTime lastActivityDate = DateTime.MinValue;
   ProfileInfoCollection profileData;
   ProfileAuthenticationOption authenticationOption;

   authenticationOption = (ProfileAuthenticationOption)
     (Convert.ToInt32(ddAuthenticationOptions.SelectedValue));

  // calculate the last activity date
  inactiveMonths = Convert.ToInt32(ddInactiveMonths.SelectedValue);
  lastActivityDate = DateTime.Now.AddMonths(-inactiveMonths);

  // get profiles matching authentication option and last activity date
  profileData =
    ProfileManager.GetAllInactiveProfiles(authenticationOption,
         lastActivityDate);

    // check to see if any profiles match the selected criteria
    if (profileData.Count == 0)
    {
   divProfiles.Visible = false;
   divMessage.Visible = true;
   litMessage.Text =
   "No profiles were found matching the entered criteria";
    }
    else
    {
   divProfiles.Visible = true;
   divMessage.Visible = false;
   gvProfiles.DataSource = profileData;
   gvProfiles.DataBind();
    }

    // store the search criteria in the ViewState to support deleting
    // profiles when required
    // NOTE: The data is stored to ensure it matches the displayed
    // profiles to handle the case where the user changes the
    // criteria and then clicks the delete all button
    ViewState[VS_AUTHENTICATION_OPTION] = authenticationOption;
    ViewState[VS_LAST_ACTIVITY_DATE] = lastActivityDate;
 } // btnSearch_Click

 ///***************************************************************************
 /// <summary>
 /// This routine provides the event handler for the delete all button
 /// click event. It is responsible for deleting the displayed profiles
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnDeleteAll_Click(Object sender,
          System.EventArgs e)
 {
    DateTime lastActivityDate;
    ProfileAuthenticationOption authenticationOption;
    int profilesDeleted;

    // get the search criteria used to display the current data
    lastActivityDate = (DateTime)(ViewState[VS_LAST_ACTIVITY_DATE]);
    authenticationOption = (ProfileAuthenticationOption)
      (ViewState[VS_AUTHENTICATION_OPTION]);

    // delete the profiles
    profilesDeleted =
   ProfileManager.DeleteInactiveProfiles(authenticationOption,
           lastActivityDate);

    // hide the grid and output message indicating the number of
    // profiles deleted

    divProfiles.Visible = false;
    divMessage.Visible = true;
    litMessage.Text = profilesDeleted.ToString() +
     " Profile(s) were deleted";
 } // btnDeleteAll_Click
  } // CH10ManagingUserProfilesCS
}
```

PREV

PREV

Recipe 10.6. Using Themes

Problem

You want to be able to change the look of your application without having to change all the pages in your application.

Solution

Use the Theme features in ASP.NET 2.0 to define the look of your application. That is, create a theme to define the look of each control and set the `Theme` attribute of the `@ Page` directive to tell ASP.NET to use the theme for controls on the page.

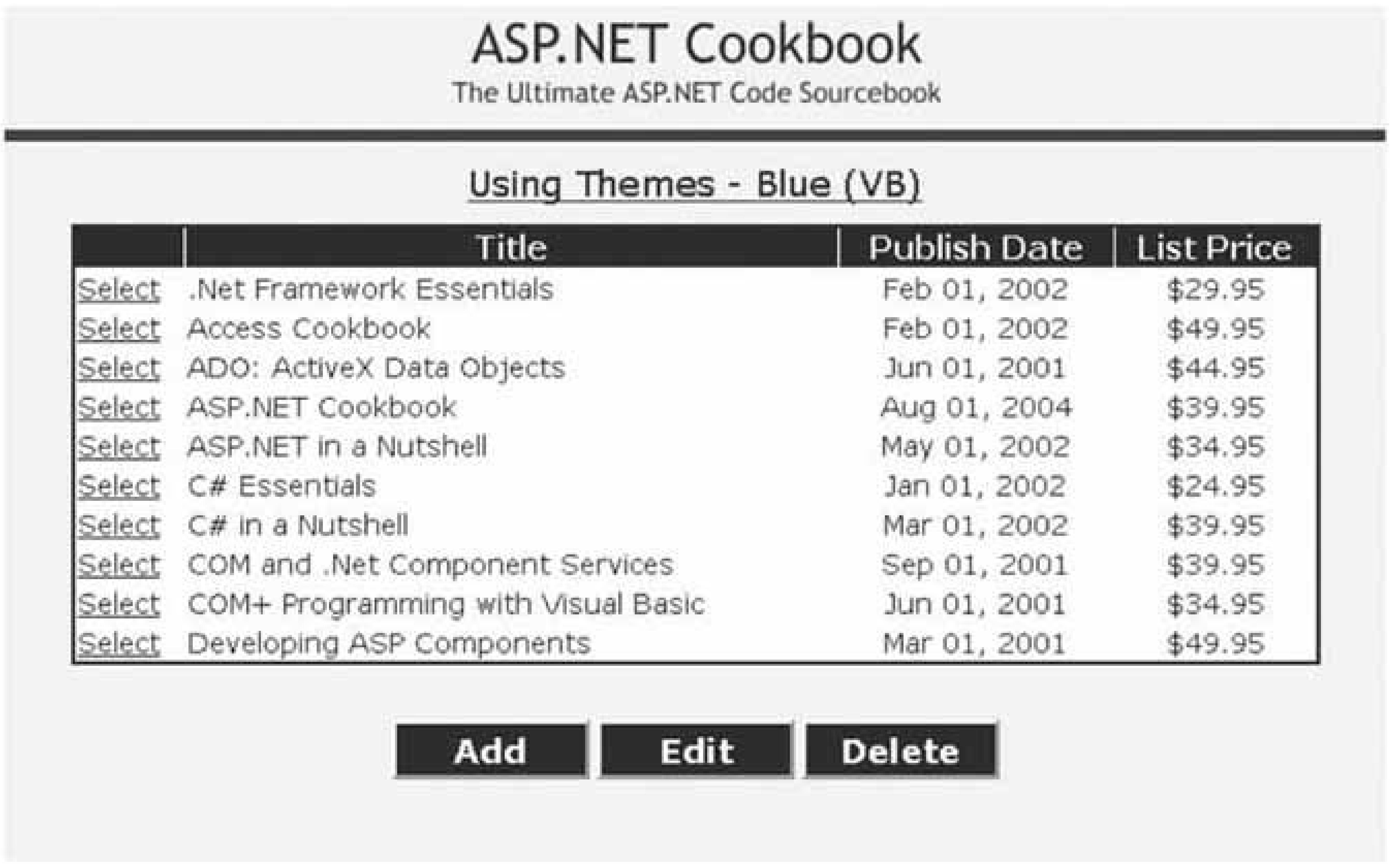
Create a theme as follows:

1. Create an `App_Themes` folder in the root of your application.
2. Create a folder with the name you want to use for the theme in the `App_Themes` folder.
3. Create a *.skin* file in the folder created above containing a definition for each server control you use in your application with the attributes that define its appearance set the way you want the control to look.
4. (Optionally) Create a *.css* file in the folder created above to use with your *.skin* file.
5. (Optionally) Add any images to the folder created above or to a subfolder as desired.

In the *.aspx* file for pages in your application set the `Theme` attribute of the `@ Page` directive to the name of the theme you want to use for the page.

The *.skin* files we created to demonstrate this solution are shown in Example 10-14 (blue theme) and Example 10-15 (grey theme). The *.aspx* file for the page that demonstrates how to use the theme is shown in Example 10-16 . The code-behind file for the page is shown in Examples 10-17 (VB) and 10-18 (C#). The output for the page is shown in Figure 10-3.

Figure 10-3. Output of demonstration page using blue theme


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Using Themes - Blue (VB)

	Title	Publish Date	List Price
Select	.Net Framework Essentials	Feb 01, 2002	$29.95
Select	Access Cookbook	Feb 01, 2002	$49.95
Select	ADO: ActiveX Data Objects	Jun 01, 2001	$44.95
Select	ASP.NET Cookbook	Aug 01, 2004	$39.95
Select	ASP.NET in a Nutshell	May 01, 2002	$34.95
Select	C# Essentials	Jan 01, 2002	$24.95
Select	C# in a Nutshell	Mar 01, 2002	$39.95
Select	COM and .Net Component Services	Sep 01, 2001	$39.95
Select	COM+ Programming with Visual Basic	Jun 01, 2001	$34.95
Select	Developing ASP Components	Mar 01, 2001	$49.95

Discussion

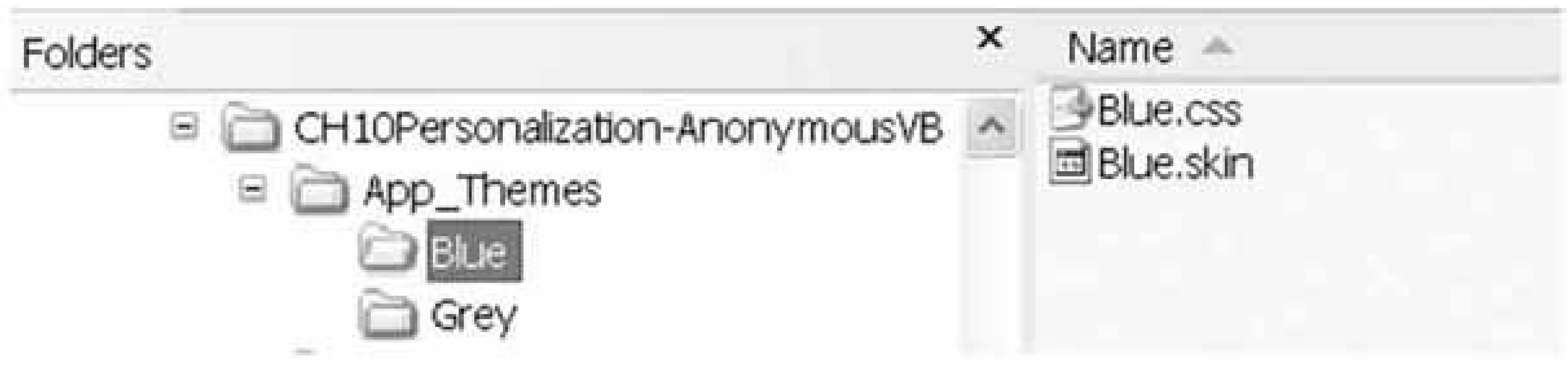
Providing a consistent look and feel for web applications has always been a challenge. In the early days of the web, all look and feel was hardcoded into the HTML in the pages. This was convenient bu had two major drawbacks. First, ensuring that all pages had a consistent look and feel required a lot of work and QA testing. Second, changing the look and feel of the application required changing every page in the application. For a large site, this was generally an expensive project.

CSS greatly diminished the problem by providing the ability to define in one place the look and feel for all aspects of the HTML page and controls. This reduced the effort of ensuring a consistent look and feel as well as the magnitude of the effort to change it. Unfortunately,style sheets can get complicated. The complexity arises from some styles inheriting from "parent" styles resulting in a confusing hierarchy of style definitions.

ASP.NET 2.0 has introduced themes to help with the problem.*Themes* are a collection of CSS files, *.skin* files, and images. Themes do not replace style sheets. Themes are intended to complement style sheets and to place all of the files associated with a given look and feel in a single location.

In ASP.NET 2.0, a special folder named`App_Themes` in the root of your application is used to store themes. Each theme needed for your application is placed in a separate folder within the`App_Themes` folder, as shown in Figure 10-4.

Figure 10-4. Theme folder structure



The *.skin* files that are part of themes contain definitions of each of the server controls used in your application along with the look and feel that is desired for the controls. The server control definitions in *.skin* files look identical to the server controls placed in the *.aspx* files, except that the only defined aspects are the visual aspectsthat is, the look and feel. An example of the`GridView` control in the *.skin* file is shown below:

```
<asp:GridView runat="server"
  BorderColor="#000080"
  BorderStyle="Solid"
  BorderWidth="2px"
  HorizontalAlign="Center"
  Width="90%">
 <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
 <RowStyle cssClass="tableCellNormal" />
 <AlternatingRowStyle cssClass="tableCellAlternating" />
 <SelectedRowStyle CssClass="tableCellSelected" />
</asp:GridView>
```

The control definitions in *.skin* files can contain an optional `SkinID` attribute to allow defining the same server control multiple times in a *.skin* file but with a different look and feel. This is useful when your application needs to use the same control in a page multiple times with different looks. When the `SkinID` attribute is not included in the control definition, the definition is used as the default definition.

```
<asp:Button runat="server"
      CssClass="button" />

    <asp:Button runat="server" SkinID="btnLightBlue"
    CssClass="lightBlueButton" />
```

A combination of server control definitions in *.skin* files and *.css* files provides the best flexibility in defining the look and feel for an application.

The simplest way to use a theme is to set the `Theme` attribute in the `@ Page` directive to the name of the theme you want applied to the page. This results in the look and feel for the default server controls defined in the *.skin* file (the ones without a SkinID) for the theme being applied to the controls on the page.

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
  CodeFile="CH10UsingThemesVB1.aspx.vb"
   Inherits="ASPNetCookbook.VBExamples.CH10UsingThemesVB1"
 Theme="Blue"
 Title="Using Themes" %>
```

To provide finer control over the application of the theme to a page, you can set the SkinID of individual controls to the SkinID of the control definition in the *.skin* file that you want applied to the control.

```
<asp:Button ID="btnEdit" runat="server" SkinID="btnLightBlue"
    Text="Edit" />
```

In the *web.config* file, you can define the theme to be used by all pages in your application. By setting the theme in *web.config* , you will not need to set the Theme attribute in the @ Page directive of each page. This approach makes globally changing to another defined theme simple.

```
<?xml  version="1.0"?>
<configuration>
  <system.web>

…

<!-
 set the theme fo all pages to to eliminate the need to add
 a Themes attribute to the @ Page directive to each page.

 -->
<pages theme="Grey" />
  </system.web>
</configuration>
```

The Theme attribute of the @ Page directive can be set in individual pages to override the setting in *web.config* for individual pages.

Themes are an excellent addition to ASP.NET 2.0, providing the ability to define and maintain a consistent look and feel. Themes can even be personalized, as described in Recipe 10.6.

See Also

Recipe 10.6

Control Definitions in .skin Files Override Definitions in .aspx Files

The attributes defined for server controls in *.skin* files override the same definitions placed in *.aspx* files. Normally, this is the behavior you want, but be careful when defining some elements of complexserver controls, such as the`GridView` . For example, it might be tempting to place the following control definition in a *.skin* file:

```
<asp:GridView runat="server"
   BorderColor="#000080"
  BorderStyle="Solid"
   BorderWidth="2px"
  HorizontalAlign="Center"
   Width="90%">
  <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
  <RowStyle cssClass="tableCellNormal" />
  <AlternatingRowStyle cssClass="tableCellAlternating" />
  <SelectedRowStyle CssClass="tableCellSelected" />
  <Columns>
 <asp:BoundField ItemStyle-HorizontalAlign="Left" />
 <asp:BoundField ItemStyle-HorizontalAlign="Center"
       DataFormatString="{0:MMM dd, yyyy}"/>
 <asp:BoundField ItemStyle-HorizontalAlign="Center"
       DataFormatString="{0:C2}"/>
  </Columns>
</asp:GridView>
```

This would seem to be all right since only look-and-feel attributes are defined. The problem arises, however, with the definition of the`<Columns>` element. The `<asp:BoundField>` elements will override the same elements in the control definition in the *.aspx* file. This problem results in a `GridView` containing no data because the `<asp:BoundField>` elements do not contain `DataField` attributes that define what data from a dataset gets bound to the column.

Example 10-14. Blue .skin file

```
<asp:GridView runat="server"
     BorderColor="#000080"
    BorderStyle="Solid"
     BorderWidth="2px"
    HorizontalAlign="Center"
     Width="90%">
 <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
 <RowStyle cssClass="tableCellNormal" />
 <AlternatingRowStyle cssClass="tableCellAlternating" />
 <SelectedRowStyle CssClass="tableCellSelected" />
</asp:GridView>

<asp:Button runat="server"
    CssClass="button" />

<asp:Button runat="server" SkinID="btnLightBlue"
    CssClass="lightBlueButton" />
```

Example 10-15. Gray .skin file

```
<asp:GridView runat="server"
     BorderColor="#C0C0C0"
    BorderStyle="Solid"
     BorderWidth="2px"
    HorizontalAlign="Center"
     Width="90%">
 <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
 <RowStyle cssClass="tableCellNormal" />
 <AlternatingRowStyle cssClass="tableCellAlternating" />
 <SelectedRowStyle CssClass="tableCellSelected" />
</asp:GridView>

<asp:Button runat="server"
     CssClass="button" />
```

Example 10-16. Page using the blue theme (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH10UsingThemesVB1.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH10UsingThemesVB1"
 Theme="Blue"
 Title="Using Themes" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
   Using Themes - Blue (VB)
 </div>
 <asp:GridView ID="gvBooks" Runat="Server"
    AllowPaging="false"
    AllowSorting="false"
    AutoGenerateColumns="false"
    Caption=""
    EnableTheming="true" >
 <Columns>
   <asp:ButtonField ButtonType="Link"
    CommandName="Select"
    Text="Select" />
   <asp:BoundField HeaderText="Title"
    DataField="Title"
    ItemStyle-HorizontalAlign="Left" />
   <asp:BoundField HeaderText="Publish Date"
    DataField="PublishDate"
    ItemStyle-HorizontalAlign="Center"
     DataFormatString="{0:MMM dd, yyyy}"/>
   <asp:BoundField HeaderText="List Price"
    DataField="ListPrice"
    ItemStyle-HorizontalAlign="Center"
     DataFormatString="{0:C2}"/>

  </Columns>
 </asp:GridView>
 <br />
 <table width="40%" border="0" align="center">
   <tr>
     <td align="center">
    <asp:Button ID="btnAdd" runat="server"
     Text="Add" />
  </td>
     <td align="center">
       <asp:Button ID="btnEdit" runat="server"
       Text="Edit" />
     </td>
     <td align="center">
       <asp:Button ID="btnDelete" runat="server"
       Text="Delete" />
     </td>
  </tr>
   </table>
</asp:Content>
```

Example 10-17. Page using the blue theme code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports Microsoft.VisualBasic
Imports System.Configuration.ConfigurationManager
Imports System.Data

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH10UsingThemesVB1.aspx
 ''' </summary>
 Partial Class CH10UsingThemesVB1
   Inherits System.Web.UI.Page
   '''*************************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the page load event. It
   ''' is responsible for initializing the controls on the page.
   ''' </summary>
   '''
   ''' <param name="sender">Set to the sender of the event</param>
   ''' <param name="e">Set to the event arguments</param>
   Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
     Dim dSource As SqlDataSource = Nothing
  Dim dataKeys(0) As String

  If (Not Page.IsPostBack) Then
    'configure the data source to get the data from the database
    dSource = New SqlDataSource()

    dSource.ConnectionString = _
     ConnectionStrings("dbConnectionString").ConnectionString
    dSource.DataSourceMode = SqlDataSourceMode.DataReader
    dSource.ProviderName = "System.Data.OleDb"
    dSource.SelectCommand = "SELECT TOP 10 " & _
       "BookID,Title,PublishDate,ListPrice " & _
       "FROM Book " & _
       "ORDER BY Title"

    'set the source of the data for the gridview control and bind it
    dataKeys(0) = "BookID"
    gvBooks.DataKeyNames = dataKeys
    gvBooks.DataSource = dSource
    gvBooks.DataBind()
  End If
 End Sub 'Page_Load

 End Class 'CH10UsingThemesVB1
End Namespace
```

Example 10-18. Page using the blue theme code-behind (.cs)

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

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH10UsingThemesCS1.aspx
 /// </summary>
 public partial class CH10UsingThemesCS1 : System.Web.UI.Page
 {
   ///**********************************************************************
   /// <summary>
   /// This routine provides the event handler for the page load event.
   /// It is responsible for initializing the controls on the page.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void Page_Load(object sender, EventArgs e)
   {
  SqlDataSource dSource = null;
  String[] dataKeys;

  if (!Page.IsPostBack)
  {
    // configure the data source to get the data from the database
    dSource = new SqlDataSource();

    dSource.ConnectionString = ConfigurationManager.
     ConnectionStrings["dbConnectionString"].ConnectionString;
    dSource.DataSourceMode = SqlDataSourceMode.DataReader;
    dSource.ProviderName = "System.Data.OleDb";
    dSource.SelectCommand = "SELECT TOP 10 " +
       "BookID,Title,PublishDate,ListPrice " +
       "FROM Book " +
       "ORDER BY Title";

    // set the source of the data for the gridview control and bind it
    dataKeys = new string[1] { "BookID" };
    gvBooks.DataKeyNames = dataKeys;
    gvBooks.DataSource = dSource;
    gvBooks.DataBind();
    }
  } // Page_Load
  } // CH10UsingThemesCS1
}
```

PREV

PREV

Recipe 10.7. User-Personalized Themes

Problem

You want to provide to users of your application the ability to choose how the application looks.

Solution

Use the Profile features along with the Theme features in ASP.NET 2.0. Implement the profile feature described in Recipe 10.1, modify *web.config* to add a property to store the selected theme in the profile, and set the selected theme in the `Page_PreInit` event handler of the pages in your application.

Modify *web.config* as follows:

- Add an `<add>` element to the `<properties>` element to include the name of the user's selected theme in the profile.

In the code-behind class for the pages in your application, use the .NET language of your choice as follows:

- Implement the event handler for the `Page_PreInit` event to set the theme for the page from the user's profile.

The solution we have implemented to demonstrate the solution is shown in Examples 10-19, 10-20, 10-21 through 10-22. Example 10-19 shows the modifications made to *web.config*. Example 10-20 shows the *.aspx* file used to enter the user profile data, and Examples 10-21 (VB) and 10-22 (C#) show the code-behind classes for the page used to enter the user profile data.

Discussion

In some cases, you'll want to allow the user to choose the look and feel she wants for your application. The combination of the profile and theme features in ASP.NET 2.0 provides the ability to implement this functionality in your application with very little code.

For our example, we built on the solution provided in Recipe 10.1, adding the ability to enter and sto the user-selected theme. This requires adding a property to the profile defined in *web.config* to store the name of the selected theme.

```
<?xml version="1.0"?>
```

```
<configuration>
  <system.web>

 …

 <profile enabled="true"
   defaultProvider="AspNetSqlProfileProvider"
   automaticSaveEnabled="false" >
   <providers>
     <remove name="AspNetSqlProfileProvider" />
  <add name="AspNetSqlProfileProvider"
   type="System.Web.Profile.SqlProfileProvider"
   connectionStringName="sqlConnectionString"
   applicationName="/AnonymousVB" />
   </providers>

   <properties>
     <add name="FirstName"
   type="System.String"
   serializeAs="String"
   allowAnonymous="true"
   provider="AspNetSqlProfileProvider"
   defaultValue=""
   readOnly="false" />

 …

  <add name="Theme"
   type="System.String"
   serializeAs="String"
   allowAnonymous="true"
   provider="AspNetSqlProfileProvider"
   defaultValue=""
   readOnly="false" />
 …

  </properties>
  </profile>
  </system.web>
</configuration>
```

To provide the ability for the user to select the desired theme, we have added a dropdown to the *.aspx* file that is populated in the code-behind with a list of the folders within the `App_Themes` folder. The names of these folders are the available themes in the application.

```
get list of files in the images directory (just for example here)
themes = Directory.GetDirectories(Server.MapPath("App_Themes"))
```

```
'for display purposes, remove the path to the theme
For index = 0 To themes.Length - 1
  themes(index) = New FileInfo(themes(index)).Name
Next index

'bind the list of themes to the dropdown on the form
ddThemes.DataSource = themes
ddThemes.DataBind()
```

```
// get list of files in the images directory (just for example here)
themes = Directory.GetDirectories(Server.MapPath("App_Themes"));

// for display purposes, remove the path to the theme
for (int index = 0; index < themes.Length; index++)
{
  themes[index] = new FileInfo(themes[index]).Name;
}

// bind the list of themes to the dropdown on the form
ddThemes.DataSource = themes;
ddThemes.DataBind();
```

To apply the user's selected theme to a page at runtime, the `Theme` property of the `Page` object must be set in the `Page_PreInit` event handler. This is required because the theme information is loaded immediately after the `Page_PreInit` event. Any attempt to set the `Theme` property after the `Page_PreInit` event occurs will result in an `InvalidOperationException` being thrown.

Since it is possible for no theme to be set in the user's profile, you need to verify that data is available and set a default value if the data is not available:

```
If (Profile.Theme.Length = 0) Then
  'user has not selected a theme so set the default
  Page.Theme = DEFAULT_THEME
Else
  'user has selected a theme so use it
  Page.Theme = Profile.Theme
End If
```

```
if (Profile.Theme.Length == 0)
{
  // user has not selected a theme so set the default
  Page.Theme = DEFAULT_THEME;
}
else
```

```
{
  // user has selected a theme so use it
  Page.Theme = Profile.Theme;
}
```

In our application, we provide the ability for the user to select a theme and apply it to the page. This presents a bit of a problem since, when the `Page_PreInit` event fires, the server controls on the page have not been initialized. Therefore, we cannot obtain the selected theme from the drop-down contro as we would in most any other circumstance. This requires us to fall back on the approach used in classic ASP for getting the form data posted to the server. For those of you who do not remember those days, you must obtain the value from the `Page.Request.Form` collection.

Generally, all that would normally be required to get the value is to use the ID of the drop-down control as the key for the `Form` collection:

```
'THIS WILL NOT WORK WHEN MASTER PAGES ARE USED
Page.Theme = Page.Request.Form("ddThemes").ToString()
```

In our application, we are using a master page, which complicates directly accessing the data in the `Form` collection. To guarantee unique names for controls on the rendered page, ASP.NET sets the name of the control to the unique name property of the server control. The name is a combination of the server control's ID and the content control's ID on the form, such as `ctl00$PageBody$ddThemes`. At first glance, it would appear you could use the `UniqueID` property of the themes drop-down control to access the `Form` data. Unfortunately, the server controls are not initialized, so the `UniqueID` property is unavailable.

To work around the problem, we write the value of the `UniqueID` property of the themes drop-down control to the rendered page in a hidden control when the form is initialized:

```
Private Const DD_THEMES_UNIQUE_ID As String = "ddThemeUniqueID"

…

ClientScript.RegisterHiddenField(DD_THEMES_UNIQUE_ID, _
        ddThemes.UniqueID)

private const String DD_THEMES_UNIQUE_ID = "ddThemeUniqueID";

…

ClientScript.RegisterHiddenField(DD_THEMES_UNIQUE_ID,
        ddThemes.UniqueID);
```

Because we are explicitly outputting the hidden control with a name of our choosing, we can get the stored value and use it to access the selected value for the themes drop-down control:

VB

```
ddThemesUniqueID = Page.Request.Form(DD_THEMES_UNIQUE_ID).ToString()
Page.Theme = Page.Request.Form(ddThemesUniqueID).ToString()
```

C#

```
ddThemesUniqueID = Page.Request.Form[DD_THEMES_UNIQUE_ID].ToString();
Page.Theme = Page.Request.Form[ddThemesUniqueID].ToString();
```

The workaround we described above is only necessary when a form is used to input the user's theme selection and to display the same form using the selected theme. When applying the theme to other pages, this workaround is unnecessary.

To simplify the pages in your application, do the following:

1. Create a base class that inherits from `System.Web.UI.Page`.
2. Add a `Page_PreInit` event handler to set the `Page.Theme` property from the theme in the user's profile.
3. Change the pages in your application to inherit from this new base class instead of the normal `System.Web.UI.Page`.

Using this approach allows you to implement the setting of the theme in one place for all your application pages.

See Also

Recipe 10.1

Example 10-19. Modifications to web.config to store theme name in the profile

```
<?xml version="1.0"?>
<configuration>
 <system.web>

 …

 <profile enabled="true"
    defaultProvider="AspNetSqlProfileProvider"
    automaticSaveEnabled="false" >
  <providers>
   <remove name="AspNetSqlProfileProvider" />
   <add name="AspNetSqlProfileProvider"
    type="System.Web.Profile.SqlProfileProvider"
    connectionStringName="sqlConnectionString"
    applicationName="/AnonymousVB" />
  </providers>

  <properties>
   <add name="FirstName"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
   <add name="LastName"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"

    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
   <add name="Theme"
    type="System.String"
    serializeAs="String"
    allowAnonymous="true"
    provider="AspNetSqlProfileProvider"
    defaultValue=""
    readOnly="false" />
  …

    </properties>
    </profile>
 </system.web>
</configuration>
```

Example 10-20. Update user profile data (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH10PersonalizedThemesVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH10PersonalizedThemesVB"
 Title="Personalized Themes" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
   Personalized Themes (VB)
 </div>
 <div align="center" class="subHeading">
   <br />
   <asp:Label ID="lblWelcome" runat="server" />
 </div>
 <table width="60%" align="center" border="0" cellpadding="2">
   <tr>
  <td align="right" class="labelText">FirstName: </td>
  <td><asp:TextBox ID="txtFirstName" runat="server" /></td>
   </tr>
   <tr>
  <td align="right" class="labelText">LastName: </td>
  <td><asp:TextBox ID="txtLastName" runat="server" /></td>
   </tr>
   <tr>
  <td align="right" class="labelText">Theme: </td>
  <td><asp:DropDownList ID="ddThemes" runat="server" /></td>
   </tr>
   <tr>
  <td colspan="2" align="center">
    <br />
    <asp:Button ID="btnUpdate" runat="server"
     Text="Update"
     OnClick="btnUpdate_Click" />

    </td>
   </tr>
   </table>
</asp:Content>
```

Example 10-21. Update user profile data code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.IO

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' CH10PersonalizedThemesVB.aspx
 ''' </summary>
 Partial Class CH10PersonalizedThemesVB
   Inherits System.Web.UI.Page

   'the following constant defines the name of the default theme
   Private Const DEFAULT_THEME As String = "Blue"

   'the following constant defines the name of the hidden field placed
   'in the form with the unique ID for the theme dropdown control
   Private Const DD_THEMES_UNIQUE_ID As String = "ddThemeUniqueID"

   '''*****************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the page preinit event.
   ''' It is responsible for initializing the page theme.
   ''' </summary>
   '''
   ''' <param name="sender">Set to the sender of the event</param>
   ''' <param name="e">Set to the event arguments</param>
   Protected Sub Page_PreInit(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.PreInit
  Dim ddThemesUniqueID As String

  'get the selected theme
  If (Page.IsPostBack) Then
    'page is being posted back so new theme is available in the form data
    'NOTE: The theme must be extracted from the form data because the
    '    server controls have not been initialized with the ViewState
    '    data yet and the theme must be set in the Page PreInit event.
    '
    '    Since the names of the server controls and thus the keys in the
    '    Form name/value collection are set to unique identifiers, the
    '      form variables cannot be accessed using the ID values we
    '      assigned. Because of this we have output a hidden HTML input
    '      to the form with the unique ID we need to access the Form
    '      variable.

    ddThemesUniqueID = Page.Request.Form(DD_THEMES_UNIQUE_ID).ToString()
    Page.Theme = Page.Request.Form(ddThemesUniqueID).ToString()
  Else
    'page is being initially displayed so use the theme setting from the
    'profile if it exists
   If (Profile.Theme.Length = 0) Then
     'user has not selected a theme so set the default
     Page.Theme = DEFAULT_THEME
      Else
     'user has selected a theme so use it
     Page.Theme = Profile.Theme
    End If
 End If
  End Sub 'Page_PreInit

  '''*****************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
 If (Not Page.IsPostBack) Then
   initializeForm( )
 End If
  End Sub 'Page_Load

  '''*****************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the update button click
  ''' event. It is responsible for updating the user profile from the data
  ''' on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnUpdate_Click(ByVal sender As Object, _
     ByVal e As System.EventArgs)
 Profile.FirstName = txtFirstName.Text
 Profile.LastName = txtLastName.Text
 Profile.Theme = ddThemes.SelectedItem.Text
 Profile.Save( )

 initializeForm( )
  End Sub 'btnUpdate_Click

  '''*****************************************************************
  ''' <summary>
  ''' This routine updates the form with data from the user's profile

  ''' </summary>
  Private Sub initializeForm( )
 Dim themes( ) As String
 Dim index As Integer

 txtFirstName.Text = Profile.FirstName
 txtLastName.Text = Profile.LastName
```

```
 'get list of files in the images directory (just for example here)
 themes = Directory.GetDirectories(Server.MapPath("App_Themes"))

 'for display purposes, remove the path to the theme
 For index = 0 To themes.Length - 1
   themes(index) = New FileInfo(themes(index)).Name
 Next index

 'bind the list of themes to the dropdown on the form
 ddThemes.DataSource = themes
 ddThemes.DataBind( )

 'select the user's selected theme
 'NOTE: Page.Theme was initialized to the correct theme in Page_PreInit
 ddThemes.SelectedIndex = _
   ddThemes.Items.IndexOf(ddThemes.Items.FindByValue(Page.Theme))

 'output the unique identifier for the themes dropdown list to a hidden
 'field. The need for this is explained in the Page_PreInit event handler
 ClientScript.RegisterHiddenField(DD_THEMES_UNIQUE_ID, _
       ddThemes.UniqueID)
   End Sub 'initializeForm
  End Class 'CH10PersonalizedThemesVB
End Namespace
```

Example 10-22. Update user profile data code-behind (.cs)

```
using System;
using System.Configuration;
using System.IO;
using System.Web.UI.WebControls;
using System.Web.Profile;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 /// CH10PersonalizedThemesCS.aspx
 /// </summary>
 public partial class CH10PersonalizedThemesCS : System.Web.UI.Page
 {

   // the following constant defines the name of the default theme
   private const String DEFAULT_THEME = "Blue";

   // the following constant defines the name of the hidden field placed
   // in the form with the unique ID for the theme dropdown control
   private const String DD_THEMES_UNIQUE_ID = "ddThemeUniqueID";

   ///*****************************************************************
   /// <summary>
   /// This routine provides the event handler for the page preinit event.
   /// It is responsible for initializing the page theme.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void Page_PreInit(Object sender,
         System.EventArgs e)
   {
  String ddThemesUniqueID;

  // get the selected theme
  if (Page.IsPostBack)
  {
    // page is being posted back so new theme is available in the form data
    // NOTE: The theme must be extracted from the form data because the
    //  server controls have not been initialized with the ViewState
    //  data yet and the theme must be set in the Page PreInit event.
    //
    //  Since the names of the server controls and thus the keys in
    //  the Form name/value collection are set to unique identifiers,
    //  the form variables cannot be accessed using the ID values we
    //  assigned. Because of this we have output a hidden HTML input
    //  to the form with the unique ID we need to access the Form
    //  variable.
    ddThemesUniqueID = Page.Request.Form[DD_THEMES_UNIQUE_ID].ToString( );
    Page.Theme = Page.Request.Form[ddThemesUniqueID].ToString( );
  }
  else
  {
    // page is being initially displayed so use the theme setting from the
    // profile if it exists
    if (Profile.Theme.Length == 0)
    {
   // user has not selected a theme so set the default
   Page.Theme = DEFAULT_THEME;
    }
    else
    {
   // user has selected a theme so use it
   Page.Theme = Profile.Theme;
    }
  }
 } // Page_PreInit

 ///*****************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
```

```
/// It is responsible for initializing the controls on the page.
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void Page_Load(object sender, EventArgs e)
{
 if (!Page.IsPostBack)
 {
   initializeForm( );
 }
} // Page_Load

///***************************************************************
/// <summary>
/// This routine provides the event handler for the update button click
/// event. It is responsible for updating the user profile from the data
/// on the page.
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void btnUpdate_Click(Object sender, System.EventArgs e)
{
 Profile.FirstName = txtFirstName.Text;
 Profile.LastName = txtLastName.Text;
 Profile.Theme = ddThemes.SelectedItem.Text;
 Profile.Save( );

 initializeForm( );
} // btnUpdate_Click

///***************************************************************
/// <summary>
/// This routine updates the form with data from the user's profile
/// </summary>
private void initializeForm( )
{
 String[] themes;

 txtFirstName.Text = Profile.FirstName;
 txtLastName.Text = Profile.LastName;

 // get list of files in the images directory (just for example here)
 themes = Directory.GetDirectories(Server.MapPath("App_Themes"));

 // for display purposes, remove the path to the theme
 for (int index = 0; index < themes.Length; index++)
 {

 themes[index] = new FileInfo(themes[index]).Name;
  }
```

```
    // bind the list of themes to the dropdown on the form
    ddThemes.DataSource = themes;
    ddThemes.DataBind( );

    // select the user's selected theme
    // NOTE: Page.Theme was initialized to the correct theme in Page_PreInit
    ddThemes.SelectedIndex =
   ddThemes.Items.IndexOf(ddThemes.Items.FindByValue(Page.Theme));

    // output the unique identifier for the themes dropdown list to a hidden
    // field. The need for this is explained in the Page_PreInit event
    // handler.
    ClientScript.RegisterHiddenField(DD_THEMES_UNIQUE_ID,
          ddThemes.UniqueID);
  } // initializeForm
   } // CH10PersonalizedThemesCS
}
```

PREV

Chapter 11. Web Parts

PREV

PREV

11.0 Introduction

Web parts are the new building blocks of personalization in ASP.NET 2.0. Any controlwhether a standard server control, custom control, user control, or web part controlcan be used as a web part without modification.

In its simplest form, a web part consists of an ASP.NET server or user control that takes advantage of the *Web Parts control set*, which is a group of structural components consisting of the following minimum set:

`WebPartManager` control

Responsible for managing all other web part controls on the page

`WebPartZone` control

Defines an area on a page where web parts can be placed

`CatalogZone` control

Responsible for managing the user interface that displays the available web parts and provides the user the ability to select web parts and add them to`WebPartZones`

Creating a web part and using the Web Parts control set on an ASP.NET page is a natural first step in learning how to build web parts and is the subject of this chapter's first recipe.

As you create a stable of web parts, you will find you want to reuse them on many pages in your applications, yet having to declare each web part on each page can be a stumbling block. Creating a reusable web parts catalog circumvents this issue and is the subject of the chapter's second recipe.

ASP.NET server controls and user controls can be effective when used as web parts. Nevertheless, you may need additional functionality not provided by these controlsfor example, when you want the ability to build your web parts into a separate assembly for sharing with other applications. When thi is the case, creating a custom web part can be a good alternative and is the subject of the chapter's third recipe.

Communicating between web parts is another common scenario that you may need to support in your applications and is the subject of the fourth recipe in the chapter. For example, this recipe shows you how to communicate between one web part that acts as a filter for data to be displayed b another web part.

When you've built your own web parts, complete with custom properties, you'll want to store the custom property settings so the next time users revisit your application, their property settings will be reflected. Persisting the user's web part property settings along with the other web part personalization data is the subject of the last recipe in the chapter.

Though some of these recipes are fairly detailed, this chapter only scratches the surface of web parts which, along with web portal development, is a career topic. Having said that, once you are familiar with the basics of web parts, you will find that the ASP.NET 2.0 web part infrastructure meets the needs of most of your applications right out of the box. Indeed, you will probably have to add little code to begin implementing a portal-style application.

PREV

Recipe 11.2. Using Server Controls and User Controls as Web Parts

Problem

You want to take advantage of web part functionality while leveraging standard ASP.NET server controls or perhaps your existing user controls.

Solution

Use the ASP.NET 2.0 membership and web part features by implementing the membership features described in Recipe 9.5 and modify *web.config* to configure the web part provider. In the pages that use web parts, add a `WebPartManager` control, add one or more `WebPartZone` controls, add a `CatalogZone` control, and add the desired server controls and user controls to the `CatalogZone` .

Add a `<webParts>` element to *web.config* as follows:

```
<webParts>
 <personalization defaultProvider="AspNetSqlPersonalizationProvider">
  <providers>
   <remove name="AspNetSqlPersonalizationProvider"/>
   <add name="AspNetSqlPersonalizationProvider"
   type="System.Web.UI.WebControls.WebParts.SqlPersonalizationProvider"
   connectionStringName="sqlConnectionString"
    applicationName="CH11ExamplesVB" />
  </providers>
  <authorization>
   <deny users="*" verbs="enterSharedScope" />
   <allow users="*" verbs="modifyState" />
  </authorization>
 </personalization>
</webParts>
```

In the *.aspx* file for the pages that will use web parts:

1. Add a `WebPartManager` control.
2. Add one or more `WebPartZone` controls.
3. Add a `CatalogZone` control along with `PageCatalogPart` and `DeclarativeCatalogPart` controls.

4. Add the server controls and user controls you want to use as web parts to the `DeclarativeCatalogPart` control.

5. Add a Customize button (or equivalent) to provide the ability for the user to initiate customization of the page.

6. Optionally add a Reset button (or equivalent) to provide the ability to reset all page customizations.

In the code-behind class for the pages using web parts, use the .NET language of your choice to:

1. Implement an event handler for the Customize button click event and set the `WebPartManager DisplayMode` to `CatalogDisplayMode` to allow the user to customize the page.

2. Optionally implement an event handler for the Reset button click event and call the `ResetPersonalizationState` method to reset all user customizations.

The application we have implemented to demonstrate the solution is shown in Examples 11-1, 11-2 , 11-3 , 11-4 , 11-5 , 11-6 , through 11-7 . Example 11-1 shows a user control that displays the weather for Charlottesville, Virginia. Examples 11-2, 11-3 through 11-4 show the *.aspx* and code-behind for a user control that displays a list of books from a database. Examples 11-5, 11-6 through 11-7 show the *.aspx* and code-behind for a page that demonstrates using standard controls and user controls as web parts.

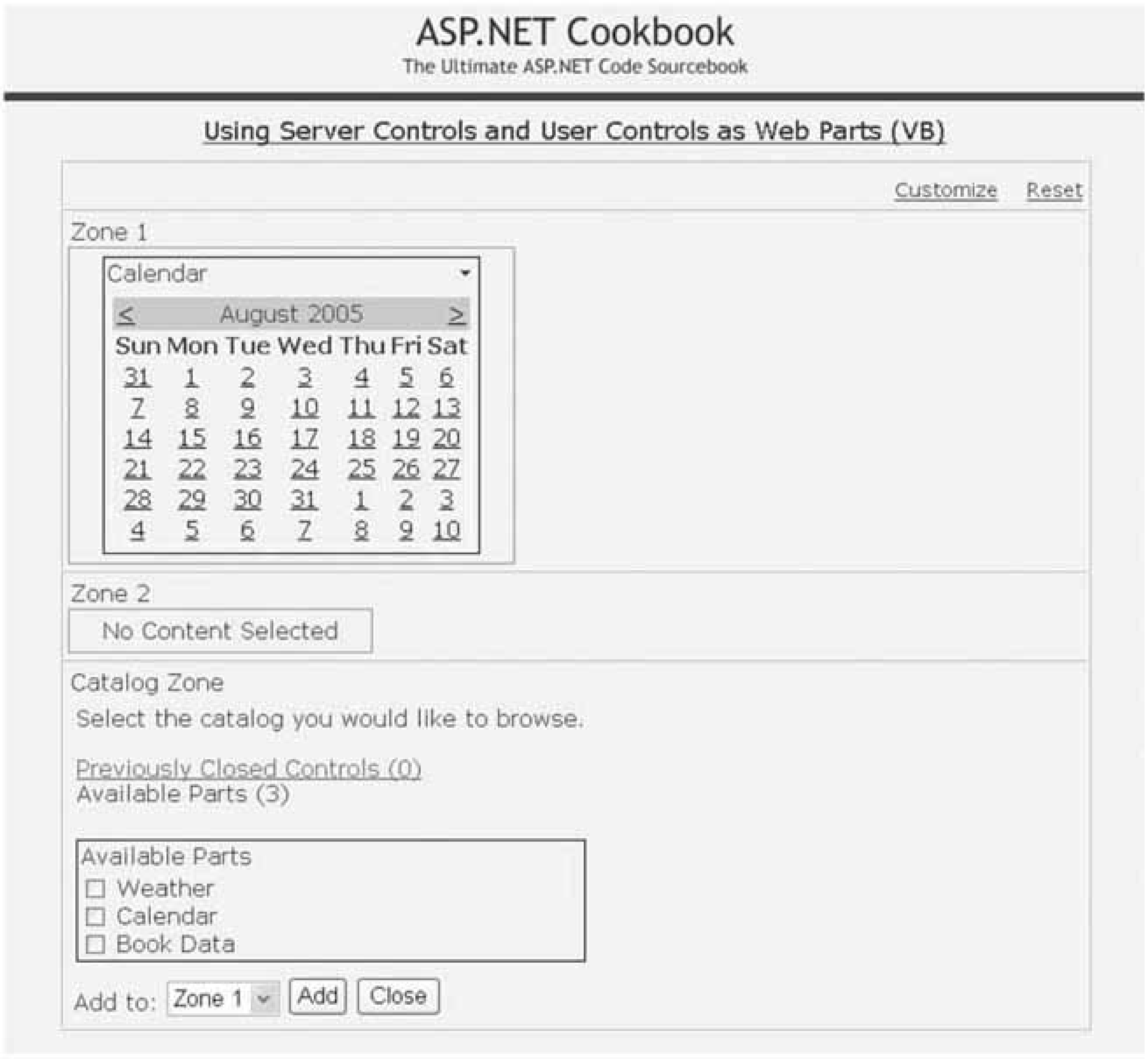
Figure 11-1 shows the demonstration page as it is originally displayed before any customization. Figure 11-2 shows the demonstration page in catalog display mode with a calendar added to the first `WebPartZone` . Figure 11-3 shows the demonstration page after customization.

Discussion

Many applications need the ability to allow users to control what content is presented on a page as well as to position the content in the location of their choosing. This functionality has been available with portal applications for many years but has been beyond the reach of most developers because o the high cost of off-the-shelf packages to support portals or the high cost of rolling your own.

Figure 11-1. Demonstration page before adding content

Figure 11-2. Demonstration page in catalog display mode



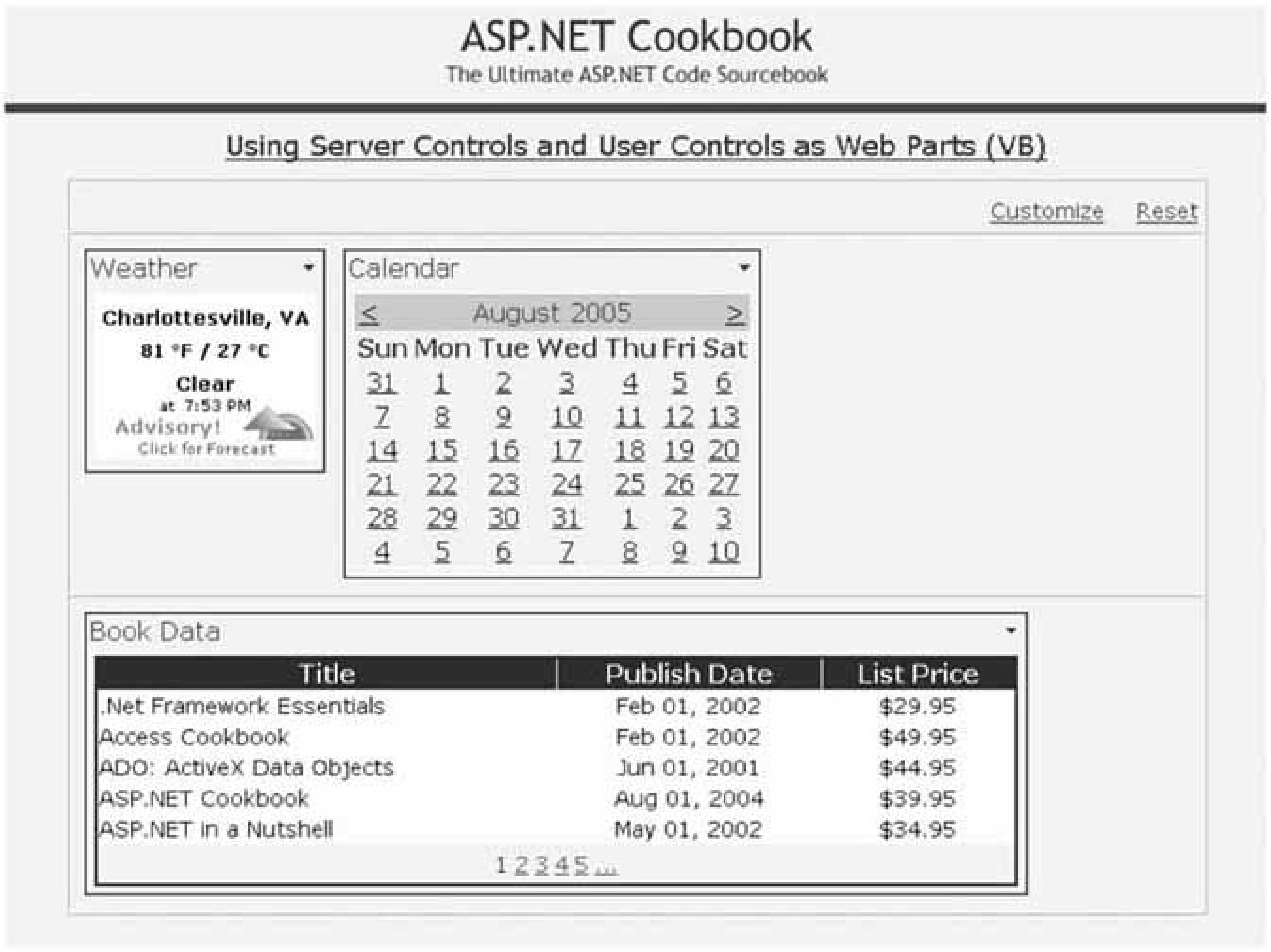
Select the catalog you would like to browse.

Available Parts

☐ Weather
☐ Calendar
☐ Book Data

ASP.NET 2.0 includes a web parts framework that provides the ability to build your own portal applications to allow users to customize the content, appearance, and behavior of pages. For all but the most complex applications, little code is required to implement a portal-style application.

Figure 11-3. Demonstration page after adding content



ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

Using Server Controls and User Controls as Web Parts (VB)

```
<  August 2005  >
Sun Mon Tue Wed Thu Fri Sat
31   1   2   3   4   5   6
 7   8   9  10  11  12  13
14  15  16  17  18  19  20
21  22  23  24  25  26  27
28  29  30  31   1   2   3
 4   5   6   7   8   9  10
```

Book Data

Title	Publish Date	List Price
.Net Framework Essentials	Feb 01, 2002	$29.95
Access Cookbook	Feb 01, 2002	$49.95
ADO: ActiveX Data Objects	Jun 01, 2001	$44.95
ASP.NET Cookbook	Aug 01, 2004	$39.95
ASP.NET in a Nutshell	May 01, 2002	$34.95

1 2 3 4 5 ...

Like most other features in ASP.NET 2.0, the web parts framework is built using the provider model. One provider for the framework is shipped with 2.0: the `SqlPersonalizationProvider` provider. Like many of the other providers, it supports using SQL Server 7 or later versions to store and retrieve the web part personalization information. If you need to use a different data store for your application, you can create your own provider.

Microsoft uses the following terminology in ASP.NET 2.0:

Membership

Describes the authentication features

Role Manager

Describes the authorization features as a function of a user's roles

Profile

Describes the features used to store information about a user

Personalization

Describes the personalization that can be done with web parts

Microsoft recognized that web parts would be more attractive to developers if they were able to use standard controls and reuse user controls that had been developed.

As part of the web part framework, support for using any standard, custom, or user control as a web part is provided. This is accomplished by automatically wrapping these "standard" controls with a `GenericWebPart` object at compilation time. This way, all of your investment in controls of all types is preserved and any of these controls can be used as web parts with no modification.

Using standard server controls and user controls as web parts may not always be the best solution, because of a few limitations on what you can do with the controls. Refer to Recipe 11.3 and Table 11-1for more information on the limitations.

In our application that implements this solution, we have implemented the`Membership` functionality for authentication and authorization, as described in Recipe 9.5, to provide the unique identification of the user for the personalization data. The use of the`Membership` features in ASP.NET 2.0 is not required to use web part personalization. What is required is to provide a unique value for each user in the `User.Identity.Name` property of the `Principal` object used for authentication, which is needed by the `SqlPersonalizationProvider` to identify the personalization data for the user. This means, for example, that the authentication solution provided in Recipe 9.1, which does not use Membership, ca be used equally well.

The Membership and Personalization providers share many tables in the database. If you want to use the Personalization feature but do not intend to use the Membership features, you still need to add the tables required for the Membership and Personalization providers. Refer to the "Using SQL Server Instead of SQLExpress with the Membership and Role Providers" sidebar in Recipe 9.5 for information on creating a database with the required tables or adding the required tables to an existing database.

Web part personalization requires authenticated users. You cannot enable personalization for anonymous users. Attempts to use personalization with anonymous users will result in an exception being thrown when the user tries to customize the page. Several different exceptions are thrown, depending on which operation is performed. All of the exception messages indicate that personalization is not enabled and/or modifiable.

Once an authentication mechanism is in place, the next step is to configure the web part framework by adding the following `<webParts>` element to *web.config* . The `connnectionStringName` must be set to the name of the connection string defined for your database in*web.config* , and `applicationName` should be set to a unique name for your application. The`applicationName` is used by the `SqlPersonalizationProvider` to identify the data for your application in the database and to allow multiple applications to share the same database while keeping the personalization data separated.

```
<webParts>
 <personalization defaultProvider="AspNetSqlPersonalizationProvider">
  <providers>
   <remove name="AspNetSqlPersonalizationProvider"/>
   <add name="AspNetSqlPersonalizationProvider"
   type="System.Web.UI.WebControls.WebParts.SqlPersonalizationProvider"
   connectionStringName="<connection string>"
   applicationName="<your application name>" />
  </providers>
  <authorization>
   <deny users="*" verbs="enterSharedScope" />
   <allow users="*" verbs="modifyState" />
  </authorization>
 </personalization>
</webParts>
```

The `<remove>` element is used to remove a previous definition of a provider with the same name. Attempts to add a provider with a name that has been defined will result in an exception being thrown. If you always remove the provider's name prior to adding a provider, you will never have to deal with the problems that can occur with colliding names in an application that uses multiple *web.config* files.

Next, you need to add the web part controls to your *.aspx* file. The first control to add is a `WebPartManger` control. The WebPartManager control is responsible for managing all other web part controls on the page. The `WebPartManager` control must be added inside the `<form>` element and before any other web part controls.

> Only one `WebPartManager` control can be placed on a page. Otherwise, a parsing error will occur.

Now you need to add one or more `WebPartZone` controls. `WebPartZone` controls define the area(s) on a page where web parts can be placed. To control the page layout better,`WebPartZones` are frequently placed in tables. In our application, we have defined a table with three rows. A`WebPartZone` control is placed in the first and second rows of the table.

The final control to add is a `CatalogZone` control. The `CatalogZone` control is responsible for managing the user interface that displays the available web parts and provides the user the ability to select wel parts and add them to `WebPartZones` . The `CatalogZone` control is not visible in the page until the `WebPartManager` is placed in the `CatalogDisplayMode` (described below). In our example we have added the `CatalogZone` control to the third row in the table.

Within the `CatalogZone` control, a `<ZoneTemplate>` element is added along with a `PageCatalogPart` control and a `DeclarativeCatalogPart` control.

The `<ZoneTemplate>` element is a container for other catalog part controls. It provides the ability to define `CatalogPart` controls declaratively.

The `PageCatalogPart` control is responsible for managing web parts that have been added to the page but have been closed by the user (more about closing web parts later). It provides the list of closed web parts and gives the user the ability to return them to a`WebPartZone` .

The `DeclarativeCatalogPart` control contains a `<WebPartsTemplate>` element that acts as a container for standard server controls, user controls, and custom web parts available to the user. The `DeclarativeCatalogPart` control provides the ability for a developer to define declaratively the controls available to the user.

Here is the complete CatalogZone control for our application:

```
<asp:CatalogZone  ID="CatalogZone1"  runat="server"
    EmptyZoneText="No Catalog Items"
    HeaderCloseVerb-Visible="false"
    CssClass="largeLabelText"
    Padding="6" >
 <ZoneTemplate>
   <asp:PageCatalogPart  ID="pcp1"  runat="server"
     Title="Previously Closed Controls" />
   <asp:DeclarativeCatalogPart  ID="dpc1"  runat="server"
     Title="Available Parts" >
    <WebPartsTemplate>
    <ASPNetCookbook:CvilleWeather ID="ucCvilleWeather"
```

```
          runat="server"
          Title="Weather" />
     <asp:Calendar ID="cal1" runat="server"
       Title="Calendar" />
    <ASPNetCookbook:BookData ID="ucBooks" runat="server"
         Title="Book Data" />
    </WebPartsTemplate>
   </asp:DeclarativeCatalogPart>
  </ZoneTemplate>
 </asp:CatalogZone>
```

In our application, we have included `Customize` and `Reset` buttons in the *.aspx* file. The `Customize` button is used to initiate the customization of the page. When the user clicks the `Customize` button, the `btnCustomize_Click` method in the code-behind is called and sets the `DisplayMode` property of the `WebPartManager` to `CatalogDisplayMode` . This causes the `CatalogZone` control to be visible on the page, which in turn provides the user the ability to add and delete web parts on the page. With Internet Explorer 5. 5 and later versions, as well as some other browsers, the user can drag web parts to different locations within a `WebPartZone` as well as between `WebPartZones` .

```
 wpm1.DisplayMode = WebPartManager.CatalogDisplayMode
```

```
 wpm1.DisplayMode = WebPartManager.CatalogDisplayMode;
```

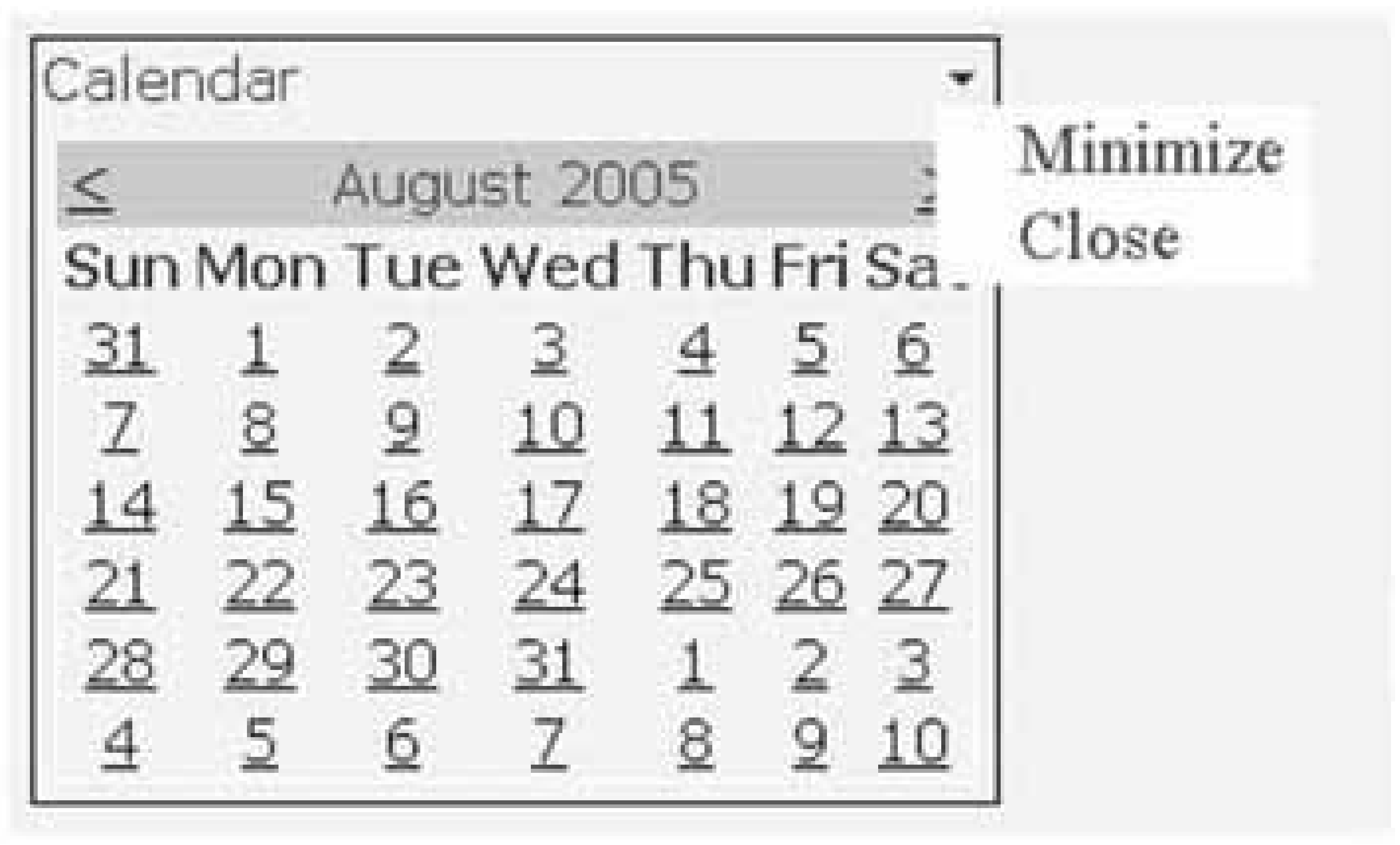
The Reset button provides the ability to reset all personalization and return the page to its original state (see Figure 11-1). When the user clicks the `Reset` button, the `btnReset_Click` method in the code-behind is called and the `ResetPersonalizationState` method of the `WebPartManager's Personalization` object is then called causing all personalization data for the page for the current user to be reset.

```
 wpm1.Personalization.ResetPersonalizationState()
```

```
wpm1.Personalization.ResetPersonalizationState();
```

When web parts are displayed in a page, a title bar is added to the control being usedas a web part. The title bar includes a title and buttons to minimize or close the web part. A calendar control used a a web part is shown in Figure 11-4.

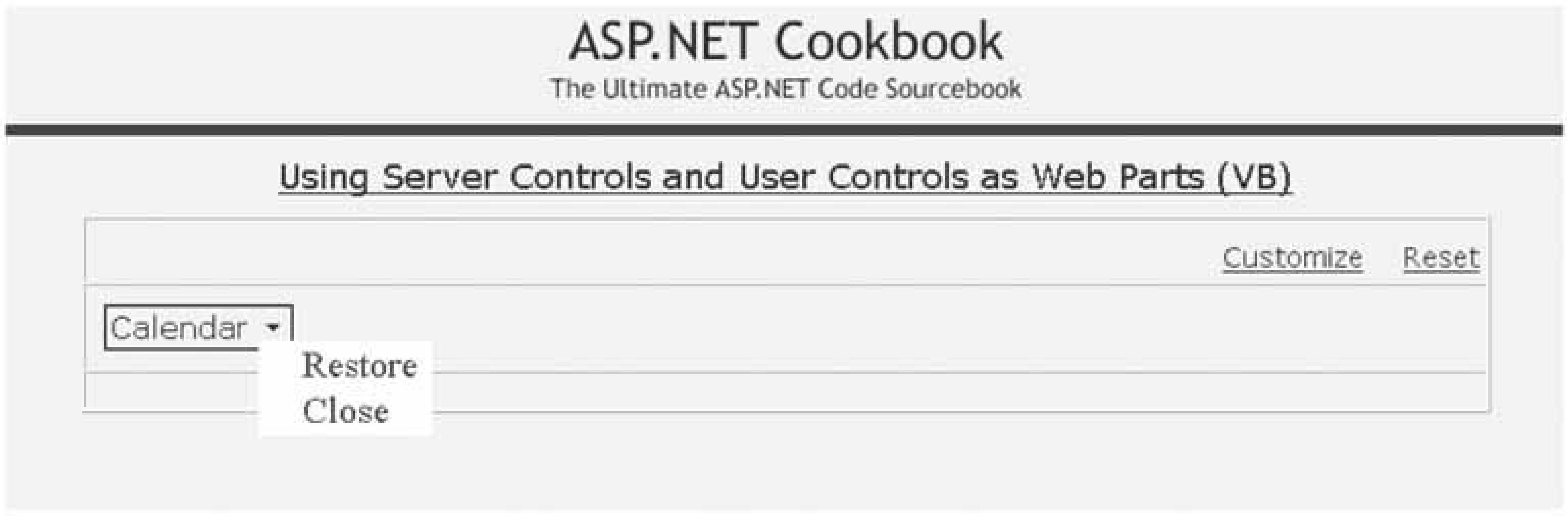
Figure 11-4. Calendar control displayed as a web part



When the `Minimize` button is clicked, the web part is collapsed on the page with only the title bar displayed, as shown in Figure 11-5. The Minimize button is then changed to a `Restore` button, as shown in Figure 11-6 .

Figure 11-5. Web part minimized

Figure 11-6. Web part minimized, showing available buttons


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Using Server Controls and User Controls as Web Parts (VB)
Customize
Reset
Calendar
Restore
Close

When the Close button is clicked, the web part is removed from the page but is not deleted. It still exists and has been added to the `PageCatalogPart` control described above. It can be added back to the page by clicking the `Customize` button, selecting the `Previously Closed Controls` catalog, selecting the desired closed control, and adding it to the desired zone.

The buttons in the web part titlebar can be displayed as a menu, as in this example, or as fixed buttons in the titlebar by setting the `WebPartVerbRenderMode` attribute of the `WebPartZone` control to `TitleBar` . The available options are `Menu` and `TitleBar` .

When the buttons are displayed as a menu, ASP.NET generates DHTML for the menu. If the requester's browser does not support the required DHTML, ASP.NET will revert to displaying the buttons in the header.

ASP.NET 2.0's web part framework provides the ability for any application to use web parts and to support portal functionality with little work. This example has only touched on some of the functionality available with web part personalization. Refer to the other recipes in this chapter as well as the `WebPartManager` class in the MSDN Library for more information on what is possible with the web part framework.

See Also

Other recipes in this chapter, Recipe 9.5, and the MSDN Library for information on the `WebPartManager` class

Example 11-1. User control to display the local weather (.ascx)

```
<%@ Control Language="VB" ClassName="CH11CVilleWeatherVB" %>
<a href="http://www.wunderground.com/US/VA/Charlottesville.html?
 bannertypeclick=infobox" target="_blank">
 <img src="http://banners.wunderground.com/weathersticker/infobox_both/
language/www/US/VA/Charlottesville.gif"
 border="0"
 alt="Click for Charlottesville, Virginia Forecast"
 height="108"
 width="144"/>
</a>
```

Example 11-2. User control to display book data (.ascx)

```
<%@ Control Language="VB" AutoEventWireup="false"
 CodeFile="CH11DisplayTabularDataVB.ascx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH11DisplayTabularDataVB"   %>
<asp:SqlDataSource ID="dSource" runat="server" />
<asp:GridView ID="gvData" Runat="Server"
     AllowPaging="true"
     AllowSorting="true"
     AutoGenerateColumns="false"
      BorderColor="#000080"
     BorderStyle="Solid"
      BorderWidth="2px" Caption=""
     HorizontalAlign="Center"
     Width="600px"
     PageSize="5"
     PagerSettings-Mode="Numeric"
      PagerSettings-PageButtonCount="5"
     PagerSettings-Position="Bottom"
     PagerStyle-HorizontalAlign="Center"
     PagerStyle-CssClass="pagerText"
     OnRowCreated="gvData_RowCreated" >
 <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
 <RowStyle cssClass="tableCellNormal" />
 <AlternatingRowStyle cssClass="tableCellAlternating" />
 <Columns>
  <asp:BoundField DataField="Title"
       HeaderText="Title "
       SortExpression="Title" />
  <asp:BoundField DataField="PublishDate"
       HeaderText="Publish Date "
       ItemStyle-HorizontalAlign="Center"
       SortExpression="PublishDate"
        DataFormatString="{0:MMM dd, yyyy}" />
  <asp:BoundField DataField="ListPrice"
```

```
       HeaderText="List Price "
       ItemStyle-HorizontalAlign="Center"
       SortExpression="ListPrice"
        DataFormatString="{0:C2}" />
 </Columns>
</asp:GridView>
```

Example 11-3. User control to display book data code-behind (.vb)

```
Option Explicit On
Option Strict On
Imports System.Configuration.ConfigurationManager
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH11DisplayTabularDataVB.ascx
 ''' </summary>
 Partial  Class  CH11DisplayTabularDataVB
  Inherits System.Web.UI.UserControl
   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   'configure the data source to get the data from the database
   'NOTE: This code must be executed anytime the page is rendered
   '      including postbacks
   dSource.ConnectionString = _
    ConnectionStrings("dbConnectionString").ConnectionString
   dSource.DataSourceMode = SqlDataSourceMode.DataSet
   dSource.ProviderName = "System.Data.OleDb"
   dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " & _
      "FROM Book " & _
      "ORDER BY Title"

   'set the data source ID for the GridView
   'NOTE: The DataSourceID must be used instead of the DataSource if the
   '      automatic sorting/paging in GridView are to be used.
   gvData.DataSourceID = dSource.ID
```

```
    If (Not Page.IsPostBack) Then
     'perform the initial sort on the first column in ascending order
      gvData.Sort(gvData.Columns(0).SortExpression, _
      SortDirection.Ascending)
    End If

  End Sub 'Page_Load

  '''**************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the GridView's row created
  ''' event. It is responsible for setting the icon in the header row to
  ''' indicate the current sort column and sort order
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub gvData_RowCreated(ByVal sender As Object, _
        ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs)
    Dim index As Integer
    Dim col As DataControlField = Nothing
    Dim image As HtmlImage = Nothing

    If (e.Row.RowType = DataControlRowType.Header) Then
     'loop through the columns in the gridview updating the header to
     'mark which column is the sort column and the sort order
     For index = 0 To gvData.Columns.Count - 1
     col = gvData.Columns(index)

     'check to see if this is the sort column
     If (col.SortExpression.Equals(gvData.SortExpression)) Then
     'this is the sort column so determine whether the ascending or
     'descending image needs to be included
     image = New HtmlImage()
     image.Border = 0
     If (gvData.SortDirection = SortDirection.Ascending) Then
     image.Src = "images/sort_ascending.gif"
     Else
     image.Src = "images/sort_descending.gif"
     End If

     'add the image to the column header
     e.Row.Cells(index).Controls.Add(image)
     End If 'If (col.SortExpression = sortExpression)
     Next index
    End If 'If (gvData.SortExpression.Equals(String.Empty))
  End Sub 'gvData_RowCreated
 End Class 'CH11DisplayTabularDataVB
End Namespace
```

Example 11-4. User control to display book data code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH11DisplayTabularDataCS.aspx
 /// </summary>
 public partial class CH11DisplayTabularDataCS : System.Web.UI.UserControl
 {
  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   // configure the data source to get the data from the database
   // NOTE: This code must be executed anytime the page is rendered
   //   including postbacks
   dSource.ConnectionString = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
   dSource.DataSourceMode = SqlDataSourceMode.DataSet;
   dSource.ProviderName = "System.Data.OleDb";
   dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " +
      "FROM Book " +
      "ORDER BY Title";

      // set the data source ID for the GridView
   // NOTE: The DataSourceID must be used instead of the DataSource if the
   //   automatic sorting/paging in GridView are to be used.
   gvData.DataSourceID = dSource.ID;

   if (!Page.IsPostBack)
   {
    // perform the initial sort on the first column in ascending order
     gvData.Sort(gvData.Columns[0].SortExpression,
    SortDirection.Ascending);
   }
  } // Page_Load

  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the GridView's row created
  /// event. It is responsible for setting the icon in the header row to
  /// indicate the current sort column and sort order
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void gvData_RowCreated(Object sender,
      System.Web.UI.WebControls.GridViewRowEventArgs e)
  {
   DataControlField col = null;
   HtmlImage image = null;

   if (e.Row.RowType == DataControlRowType.Header)
   {
    // loop through the columns in the gridview updating the header to
    // mark which column is the sort column and the sort order
    for (int index = 0; index < gvData.Columns.Count; index++)
    {
    col = gvData.Columns[index];

    // check to see if this is the sort column
    if (col.SortExpression.Equals(gvData.SortExpression))
    {
    // this is the sort column so determine whether the ascending or
    // descending image needs to be included
    image = new HtmlImage();
    image.Border = 0;
    if (gvData.SortDirection == SortDirection.Ascending)
    {
    image.Src = "images/sort_ascending.gif";
    }
    else
    {
    image.Src = "images/sort_descending.gif";
    }

    // add the image to the column header
    e.Row.Cells[index].Controls.Add(image);
    } // if (col.SortExpression.Equals(gvBooks.SortExpression))
    } // for index
   } // if (e.Row.RowType == DataControlRowType.Header)
  } //gvData_RowCreated
 } // CH11DisplayTabularDataCS
}
```

Example 11-5. Using regular controls as web parts (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH11UsingRegularContolsAsWebPartsVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH11UsingRegularContolsAsWebPartsVB"
 Title="Using Server Controls and User Controls as Web Parts" %>
<%@ Register TagPrefix="ASPNetCookbook" TagName="CvilleWeather"
     Src="~/CH11CVilleWeatherVB.ascx" %>
<%@ Register TagPrefix="ASPNetCookbook" TagName="BookData"
     Src="~/CH11DisplayTabularDataVB.ascx" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Using Server Controls and User Controls as Web Parts (VB)
 </div>
 <asp:WebPartManager ID="wpm1" runat="server" />
 <table width="90%" align="center" border="1" cellpadding="4" cellspacing="0">
  <tr>
   <td align="right">
    <asp:LinkButton ID="btnCustomize" runat="server"
        Text="Customize"
        CssClass="labelText"
        OnClick="btnCustomize_Click" />&nbsp;&nbsp;
    <asp:LinkButton ID="btnReset" runat="server"
        Text="Reset"
        CssClass="labelText"
        OnClick="btnReset_Click" />
   </td>
  </tr>
  <tr>
   <td>
    <asp:WebPartZone ID="webPartZone1" runat="server"
       EmptyZoneText="No Content Selected"
       Height="10" HeaderText="Zone 1"
       LayoutOrientation="Horizontal"
       CssClass="largeLabelText"
       Padding="6" />
   </td>
  </tr>
  <tr>
   <td>
    <asp:WebPartZone ID="webPartZone2" runat="server"
       EmptyZoneText="No Content Selected"
       Height="10" HeaderText="Zone 2"           LayoutOrientation="Horizontal"
       CssClass="largeLabelText"
       Padding="6" />
   </td>
  </tr>
  <tr>
   <td>
```

```
    <asp:CatalogZone ID="CatalogZone1" runat="server"
       EmptyZoneText="No Catalog Items"
       HeaderCloseVerb-Visible="false"
       CssClass="largeLabelText"
       Padding="6" >
    <ZoneTemplate>
    <asp:PageCatalogPart ID="pcp1" runat="server"
         Title="Previously Closed Controls" />
    <asp:DeclarativeCatalogPart ID="dpc1" runat="server"
             Title="Available Parts" >
    <WebPartsTemplate>
     <ASPNetCookbook:CvilleWeather ID="ucCvilleWeather"
              runat="server"
              Title="Weather" />
     <asp:Calendar ID="cal1" runat="server"
          Title="Calendar" />
     <ASPNetCookbook:BookData ID="ucBooks" runat="server"
            Title="Book Data" />
    </WebPartsTemplate>
    </asp:DeclarativeCatalogPart>
    </ZoneTemplate>
    </asp:CatalogZone>
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 11-6. Using regular controls as web parts code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH11UsingRegularContolsAsWebPartsVB.aspx
 ''' </summary>
 Partial  Class  CH11UsingRegularContolsAsWebPartsVB
  Inherits System.Web.UI.Page
   '''**********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the customize button
  ''' click event. It is responsible for placing the web part manager
  ''' in catalog mode.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnCustomize_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
   wpm1.DisplayMode = WebPartManager.CatalogDisplayMode
  End Sub 'btnCustomize_Click

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the reset button
  ''' click event. It is responsible for resetting the web part manager
  ''' personalization data.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnReset_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)
   wpm1.Personalization.ResetPersonalizationState()
  End Sub 'btnReset_Click
 End Class 'CH11UsingRegularContolsAsWebPartsVB
End Namespace
```

Example 11-7. Using regular controls as web parts code-behind (.cs)

```
using System;
using System.Web.UI.WebControls.WebParts;
namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH11UsingRegularContolsAsWebPartsCS.aspx
 /// </summary>
 public partial class CH11UsingRegularContolsAsWebPartsCS :
  System.Web.UI.Page
 {

 ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the customize button
 /// click event. It is responsible for placing the web part manager
 /// in catalog mode.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnCustomize_Click(Object sender,
        System.EventArgs e)
  {
   wpm1.DisplayMode = WebPartManager.CatalogDisplayMode;
  } // btnCustomize_Click

  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the reset button
  /// click event. It is responsible for resetting the web part manager
  /// personalization data.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnReset_Click(Object sender,
        System.EventArgs e)
  {
   wpm1.Personalization.ResetPersonalizationState();
  } // btnReset_Click
 } // CH11UsingRegularContolsAsWebPartsCS
}
```

PREV

Recipe 11.3. Creating a Reusable Web Parts Catalog

Problem

You need to reuse the same set of web parts in many pages in your application but you want to avoi
having to declare them on every page. Instead, you'd like to create areusable web parts catalog you can maintain separately but make available to the pages.

Solution

Implement the solution described in Recipe 11.1, create a user control containing all of the web parts to be available in your pages, remove the `<WebPartsTemplate>` element and control definitions from the `DeclarativeCatalogPart` control, and set the `WebPartsListUserControlPath` attribute of the `DeclarativeCatalogPart` to the name of the user control you created.

In the *.ascx* file for the user control that will contain all available web parts, add the server controls, user controls, and custom web parts in the same manner you would add them to a *.aspx* file.

In the *.aspx* file:

1. Remove the `<WebPartsTemplate>` element and control definitions from the `DeclarativeCatalogPart` control.
2. Set the `WebPartsListUserControlPath` attribute of the `DeclarativeCatalogPart` to the name of the user control containing the available web parts.

The application we have implemented to demonstrate the solution is shown in Examples 11-8, 11-9, 11-10 through 11-11. Example 11-8 shows the user control used as a "web parts catalog." Examples 11-9, 11-10 through 11-11 show the *.aspx* and code-behind for a page that demonstrates using the "web parts catalog" user control.

Discussion

Applications that use web parts commonly reuse the same web parts in most or all of the pages of the application. If your application only has a couple of web parts and the list never changes, declaring them in every page where they can be used is not a problem. However, if you have many web parts and new parts are periodically added, this becomes a maintenance problem since every page that uses the web parts would require revision to add the new parts.

The web parts framework provides the ability to address this problem by using a user control to define the web parts available instead of listing them individually in the DeclarativeCatalogPart control. This approach, which allows you to manage the list of available web parts in a single location significantly reduces the effort required to change the list of available web parts.

The user control that contains the available web parts is created in the same manner as any other user control, but it is used differently. It is never visible to the user, so you do not need to be concerned about control layout. It is used by the DeclarativeCatalogPart control to provide the list of available web parts.

The order in which the controls are defined in the "web part catalog" user control is the same order in which they will be presented to the user.

In our application that implements this solution, we started with the example presented in Recipe 11.1. We added a new user control that contains a declaration of each of the controls to be used as web parts. As in Recipe 11.1, a standard calendar control, the "weather" user control, and the "book list" user control are declared, as shown in Example 11-9.

Next, we remove the <WebPartsTemplate> element and control definitions from the DeclarativeCatalogPart control in the *.aspx* file. Then, we set the WebPartsListUserControlPath attribute of the DeclarativeCatalogPart control to the name of our "web parts catalog" user control (*CH11WebPartCatalogVB.ascx*). The complete CatalogZone control for our application is shown below.

```
<asp:CatalogZone  ID="CatalogZone1"  runat="server"
    EmptyZoneText="No Catalog Items"
    HeaderCloseVerb-Visible="false"
    CssClass="largeLabelText"
    Padding="6" >
  <ZoneTemplate>
    <asp:PageCatalogPart  ID="pcp1"  runat="server"
      Title="Previously Closed Controls" />
   <asp:DeclarativeCatalogPart ID="dpc1" runat="server"
    Title="Available Parts"
    WebPartsListUserControlPath="~/CH11WebPartCatalogVB.ascx" >
   </asp:DeclarativeCatalogPart>
  </ZoneTemplate>
</asp:CatalogZone>
```

The "web parts catalog" user control shown in this recipe functions exactly the same as the example in Recipe 11.1 but with the added benefit of reduced maintenance.

See Also

Recipe 11.1

Example 11-8. User control containing available web parts (.ascx)

```
<%@ Control Language="VB" ClassName="CH11WebPartCatalogVB" %>
<%@ Register TagPrefix="ASPNetCookbook" TagName="CvilleWeather"
     Src="~/CH11CVilleWeatherVB.ascx" %>
<%@ Register TagPrefix="ASPNetCookbook" TagName="BookData"
     Src="~/CH11DisplayTabularDataVB.ascx" %>
<ASPNetCookbook:CvilleWeather ID="ucCvilleWeather"
      runat="server"
      Title="Weather" />
<asp:Calendar ID="cal1" runat="server"
     Title="Calendar" />
<ASPNetCookbook:BookData ID="ucBooks" runat="server"
    Title="Book Data" />
```

Example 11-9. Using a web parts catalog (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH11UsingReusableWebPartCatalogVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH11UsingReusableWebPartCatalogVB"
 Title="Reusable Web Part Catalog" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Reusable Web Part Catalog (VB)
 </div>
 <asp:WebPartManager ID="wpm1" runat="server" Personalization-Enabled="true" />
 <table width="90%" align="center" border="1" cellpadding="4" cellspacing="0">
  <tr>
   <td align="right">
    <asp:LinkButton ID="btnCustomize" runat="server"
        Text="Customize"
        CssClass="labelText"
        OnClick="btnCustomize_Click" />&nbsp;&nbsp;
    <asp:LinkButton ID="btnReset" runat="server"
        Text="Reset"
        CssClass="labelText"
        OnClick="btnReset_Click" />
   </td>
  </tr>
  <tr>
   <td>
     <asp:WebPartZone ID="webPartZone1" runat="server"
       EmptyZoneText="No Content Selected"
```

```
        Height="10" HeaderText="Zone 1"
        LayoutOrientation="Horizontal"
        CssClass="largeLabelText"
        Padding="6" />
    </td>
   </tr>
   <tr>
    <td>
      <asp:WebPartZone ID="webPartZone2" runat="server"
        EmptyZoneText="No Content Selected"
        Height="10" HeaderText="Zone 2"
        LayoutOrientation="Horizontal"
        CssClass="largeLabelText"
        Padding="6" />
    </td>
   </tr>
   <tr>
    <td>
      <asp:CatalogZone ID="CatalogZone1" runat="server"
        EmptyZoneText="No Catalog Items"
        HeaderCloseVerb-Visible="false"
        CssClass="largeLabelText"
        Padding="6" >
     <ZoneTemplate>
      <asp:PageCatalogPart ID="pcp1" runat="server"
          Title="Previously Closed Controls" />
     <asp:DeclarativeCatalogPart ID="dpc1" runat="server"
              Title="Available Parts"
              WebPartsListUserControlPath="~/CH11WebPartCatalogVB.ascx" >
     </asp:DeclarativeCatalogPart>
     </ZoneTemplate>
     </asp:CatalogZone>
    </td>
   </tr>
  </table>
</asp:Content>
```

Example 11-10. Using a web parts catalog code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 ''' CH11UsingReusableWebPartCatalogVB.aspx
 ''' </summary>
 Partial Class CH11UsingReusableWebPartCatalogVB
  Inherits System.Web.UI.Page

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the customize button
  ''' click event. It is responsible for placing the web part manager
  ''' in catalog mode.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnCustomize_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
    wpm1.DisplayMode = WebPartManager.CatalogDisplayMode
  End Sub 'btnCustomize_Click

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the reset button
  ''' click event. It is responsible for resetting the web part manager
  ''' personalization data.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnReset_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)
    wpm1.Personalization.ResetPersonalizationState()
  End Sub 'btnReset_Click
 End Class 'CH11UsingReusableWebPartCatalogVB
End Namespace
```

Example 11-11. Using a web parts catalog code-behind (.cs)

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

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 /// CH11UsingReusableWebPartCatalogCS.aspx
 /// </summary>
 public partial class CH11UsingReusableWebPartCatalogCS : System.Web.UI.Page
 {
   ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the customize button
  /// click event. It is responsible for placing the web part manager
  /// in catalog mode.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnCustomize_Click(Object sender,
         System.EventArgs e)
  {
    wpm1.DisplayMode = WebPartManager.CatalogDisplayMode;
  } // btnCustomize_Click

   ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the reset button
  /// click event. It is responsible for resetting the web part manager
  /// personalization data.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnReset_Click(Object sender,
        System.EventArgs e)
  {
    wpm1.Personalization.ResetPersonalizationState();
  } // btnReset_Click
 } // CH11UsingReusableWebPartCatalogCS
}
```

Recipe 11.4. Creating a Custom Web Part

Problem

You need some functionality for your web parts that you cannot get with user controls or standard ASP.NET server controls, such as the ability to build them into a separate assembly for sharing with other applications.

Solution

Create a class that inherits from the `WebPart` class and implements the required functionality. Then, use the class with the Web Part control set in the same manner as you would other controls.

Use the .NET language of your choice to:

1. Create a new class inheriting from the `WebPart` class.
2. Override the `CreateChildControls` method and create the controls required for the web part.

Use the custom web part with the Web Part control set in the same manner as you would any other control (see Recipe 11.1 for details).

The application we have implemented to demonstrate the solution is shown in Examples 11-12, 11-13, 11-14, 11-15 through 11-16. Examples 11-12 (VB) and 11-13 (C#) show the class to implement the custom web part. Examples 11-14, 11-15 through 11-16 show the *.aspx* and code-behind for a page that demonstrates using the custom web part.

Discussion

For most applications, user controls and standard ASP.NET server controls provide the best option fo use as web parts. The primary reasons are their reusability and, in the case of user controls, the efficiency of developing them. On the other hand, developing custom web parts is akin to developing custom server controls (see Chapter 6). Since the entire user interface is generated programmatically, they can be time-consuming to develop and maintain. There are times, however, when a custom web part is the better solution. Table 11-1 lists some of the differences in custom web parts and user controls when used as web parts.

Table 11-1. When to create custom web parts

Feature	Custom web part	User control
Can be built into a separate assembly for sharing with other applications or installation into the GAC	Yes	No
Verbs can be extended	Yes	No
Can be installed in the Visual Studio toolbox	Yes	No
Designer support available in Visual Studio	No	Yes

In our application that implements this solution, we start with the example presented in Recipe 11.1 and add a custom web part by creating a class that inherits from the `WebPart` class. For purposes of comparison, our custom web part implements the same functionality as the user control we created in Recipe 11.1that is, to display a list of books.

In the class, we implemented a `CreateChildControls` method that overrides the `CreateChildControls` method in the `WebPart` class and creates the controls required for our web part (a `GridView` and a `SqlDataSource`).

The first step is to clear the controls collection. This is needed to ensure that only the controls that exist in the web part are the ones created in the `CreateChildControls` method.

```
Controls.Clear()
```

```
Controls.Clear();
```

The next step is to create the `GridView` control and set all of the properties required to define the user interface. As with custom server controls, all user interface definition is done programmatically.

```
gvData = New GridView()
gvData.AllowPaging = True
gvData.AllowSorting = True
gvData.AutoGenerateColumns = False
gvData.BorderColor  =  ColorTranslator.FromHtml("#000080")
gvData.BorderStyle = WebControls.BorderStyle.Solid
gvData.BorderWidth  = 2
gvData.Caption = ""
gvData.HorizontalAlign = WebControls.HorizontalAlign.Center
gvData.Width  = 600
gvData.PageSize  = 5
gvData.PagerSettings.Mode = PagerButtons.Numeric
gvData.PagerSettings.PageButtonCount  =  5
gvData.PagerSettings.Position = PagerPosition.Bottom
gvData.PagerStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
gvData.PagerStyle.CssClass = "pagerText"

gvData.HeaderStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
gvData.HeaderStyle.CssClass = "tableHeader"
gvData.RowStyle.CssClass = "tableCellNormal"
gvData.AlternatingRowStyle.CssClass = "tableCellAlternating"
```

```
gvData = new GridView();
gvData.AllowPaging = true;
gvData.AllowSorting = true;
gvData.AutoGenerateColumns = false;
gvData.BorderColor  =  ColorTranslator.FromHtml("#000080");
gvData.BorderStyle = BorderStyle.Solid;
gvData.BorderWidth  =  2;
gvData.Caption = "";
gvData.HorizontalAlign = HorizontalAlign.Center;
```

```
gvData.Width  = 600;
gvData.PageSize  = 5;
gvData.PagerSettings.Mode = PagerButtons.Numeric;
gvData.PagerSettings.PageButtonCount  =  5;
gvData.PagerSettings.Position = PagerPosition.Bottom;
gvData.PagerStyle.HorizontalAlign = HorizontalAlign.Center;
gvData.PagerStyle.CssClass = "pagerText";

gvData.HeaderStyle.HorizontalAlign = HorizontalAlign.Center;
gvData.HeaderStyle.CssClass = "tableHeader";
gvData.RowStyle.CssClass = "tableCellNormal";
gvData.AlternatingRowStyle.CssClass = "tableCellAlternating";
```

Next, you need to create the columns in the `GridView`, set the column properties, and add the columns to the `GridView`. The properties that need to be set include the data field used to populate the column, the header text, the sort expression, and optionally any formatting and styles.

```
gridColumn = New BoundField()
gridColumn.DataField = "Title"
gridColumn.HeaderText = "Title"
gridColumn.SortExpression = "Title"
gvData.Columns.Add(gridColumn)

gridColumn = New BoundField()
gridColumn.DataField = "PublishDate"
gridColumn.HeaderText = "Publish Date"
gridColumn.SortExpression = "PublishDate"
gridColumn.DataFormatString = "{0:MMM dd, yyyy}"
gridColumn.ItemStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
gvData.Columns.Add(gridColumn)

gridColumn = New BoundField()
gridColumn.DataField = "ListPrice"
gridColumn.HeaderText = "List Price"
gridColumn.SortExpression = "ListPrice"
gridColumn.DataFormatString = "{0:C2}"
gridColumn.ItemStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
gvData.Columns.Add(gridColumn)
```

```
gridColumn = new BoundField();
gridColumn.DataField = "Title";
gridColumn.HeaderText = "Title";
gridColumn.SortExpression = "Title";
gvData.Columns.Add(gridColumn);

gridColumn = new BoundField();
gridColumn.DataField = "PublishDate";
gridColumn.HeaderText = "Publish Date";
gridColumn.SortExpression = "PublishDate";
gridColumn.DataFormatString = "{0:MMM dd, yyyy}";
gridColumn.ItemStyle.HorizontalAlign = HorizontalAlign.Center;
gvData.Columns.Add(gridColumn);


gridColumn = new BoundField();
gridColumn.DataField = "ListPrice";
gridColumn.HeaderText = "List Price";
gridColumn.SortExpression = "ListPrice";
gridColumn.DataFormatString = "{0:C2}";
gridColumn.ItemStyle.HorizontalAlign = HorizontalAlign.Center;
gvData.Columns.Add(gridColumn);
```

In our example, we add an icon in the column header to indicate which is the sorted column and its sort order (see Recipe 2.14 for a full explanation of this technique). To do this, we must "wire" the event handler for the `RowCreated` event to our event handler method (`gvData_RowCreated`).

```
AddHandler gvData.RowCreated, AddressOf gvData_RowCreated
```

```
gvData.RowCreated += new GridViewRowEventHandler(gvData_RowCreated);
```

At this point, the `GridView` is complete and needs to be added to the Controls collection of our web part:

```
Me.Controls.Add(gvData)
```

```
this.Controls.Add(gvData);
```

Now, you need to create the `SqlDataSource` control that will retrieve the book data from a database:

```
dSource = New SqlDataSource()
dSource.ConnectionString = ConfigurationManager. _
 ConnectionStrings("dbConnectionString").ConnectionString
dSource.DataSourceMode = SqlDataSourceMode.DataSet
dSource.ProviderName = "System.Data.OleDb"
dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " & _
   "FROM Book " & _
   "ORDER BY Title"
dSource.ID = "dSource"
```

```
dSource = new SqlDataSource();
dSource.ConnectionString = ConfigurationManager.
 ConnectionStrings["dbConnectionString"].ConnectionString;
dSource.DataSourceMode = SqlDataSourceMode.DataSet;
dSource.ProviderName = "System.Data.OleDb";
dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " +
   "FROM Book " +
     "ORDER BY Title";
dSource.ID = "dSource";
```

The next step is to connect the `GridView` and the `SqlDataSource` to populate the `GridView` from the `SqlDataSource` during data binding. This is done by setting the `DataSourceID` of the `GridView` to the `ID` of the `SqlDataSource`.

```
gvData.DataSourceID = dSource.ID
```

```
gvData.DataSourceID = dSource.ID;
```

The GridView's built-in sorting and paging will work only if the DataSourceID property of the GridView is set to the ID of the data source. If the DataSource property is used instead, you will have to implement all of the same event handlers that are required to perform sorting and paging with a DataGrid (see Recipe 2.11).

As with the GridView, you need to add the SqlDataSource to the Controls collection of the web part:

VB

```
Me.Controls.Add(dSource)
```

C#

```
this.Controls.Add(dSource);
```

By default, the SortExpression property of the GridView is set to an empty string, and the SortDirection property is set to Ascending. To set the sort column and sort order to meet your needs, you will need to call the Sort method of the GridView, passing the desired sort expression and sort order in the process. This is required since the SortExpression and SortDirection properties are read-only and cannot be set directly. In our application, we set the initial sort column to the Title column and the sort order to Ascending.

```
If (gvData.SortExpression.Length = 0) Then
  gvData.Sort(gvData.Columns(0).SortExpression, _
    SortDirection.Ascending)
End If
```

```
if (gvData.SortExpression.Length == 0)
{
  gvData.Sort(gvData.Columns[0].SortExpression,
    SortDirection.Ascending);
}
```

The CreateChildControls method is executed when the page using the web part is initially displayed, as well as when postbacks occur. When a postback occurs as a result of a sort request, the SortExpression is set and is required to sort the data by the column the user clicked. In this scenario, the call should not be made to the Sort method in your code. If the call is made, the sorting will not work correctly.

Now that you have the custom web part implemented, you need to use it in a page, which is done in the same manner as standard controls and user controls (see Recipes 11.1 and 11.2). In our example, we replaced the book data user control in the `<WebPartsTemplate>` element of the `DeclarativeCatalogPart` with our custom web part.

VB

```
<asp:CatalogZone  ID="CatalogZone1"  runat="server"
    EmptyZoneText="No Catalog Items"
    HeaderCloseVerb-Visible="false"
    CssClass="largeLabelText"
    Padding="6" >
```

VB

```
 <ZoneTemplate>
   <asp:DeclarativeCatalogPart  ID="dpc1"  runat="server"
          Title="Available Parts" >
    <WebPartsTemplate>
    <ASPNetCookbook:CvilleWeather ID="ucCvilleWeather"
           runat="server"
           Title="Weather" />
     <asp:Calendar  ID="cal1"  runat="server"
       Title="Calendar" />
    <ASPNetCookbook:CH11 CustomWebPartVB ID="cwpBooks" runat="server"
                Title="Book Data" />
    </WebPartsTemplate>
   </asp:DeclarativeCatalogPart>
 </ZoneTemplate>
</asp:CatalogZone>
```

```
<asp:CatalogZone  ID="CatalogZone1"  runat="server"
    EmptyZoneText="No Catalog Items"
    HeaderCloseVerb-Visible="false"
    CssClass="largeLabelText"
    Padding="6" >
 <ZoneTemplate>
   <asp:DeclarativeCatalogPart  ID="dpc1"  runat="server"
          Title="Available Parts" >
    <WebPartsTemplate>
    <ASPNetCookbook:CvilleWeather ID="ucCvilleWeather"
           runat="server"
           Title="Weather" />
     <asp:Calendar  ID="cal1"  runat="server"
       Title="Calendar" />
    <ASPNetCookbook:CH11CustomWebPartCS ID="cwpBooks" runat="server"
              Title="Book Data" />
    </WebPartsTemplate>
   </asp:DeclarativeCatalogPart>
 </ZoneTemplate>
</asp:CatalogZone>
```

This recipe provides a simple example of a custom web part. Refer to Recipe 11.4, Recipe 11.5, and the `WebPart` class documentation in the MSDN Library for more information on other things you can do with custom web parts.

See Also

Recipes 2.11, 2.14, 11.1, 11.2, 11.4, 11.5, and the `WebPart` documentation in the MSDN Library

Example 11-12. Custom web part control (.vb)

```
Option Explicit On
Option Strict On

Imports System.Drawing
Imports System.Configuration
Imports System.Data
Imports System.Data.SqlClient

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides a custom web part that displays book data
 ''' </summary>
 Public Class CH11CustomWebPartVB
  Inherits WebPart

  Private gvData As GridView

  '''*********************************************************************
  ''' <summary>
  ''' This routine creates the child controls (GridView and SqlDataSource)
  ''' for the web part
  ''' </summary>

  Protected Overrides Sub CreateChildControls()
   Dim gridColumn As BoundField
   Dim dSource As SqlDataSource

   'clear the controls collection
   Controls.Clear()

   'create the GridView control and set the applicable properties
   'to display the book data
   gvData = New GridView()
   gvData.AllowPaging = True
   gvData.AllowSorting = True
   gvData.AutoGenerateColumns = False
   gvData.BorderColor = ColorTranslator.FromHtml("#000080")
   gvData.BorderStyle = WebControls.BorderStyle.Solid
   gvData.BorderWidth = 2
   gvData.Caption = ""
   gvData.HorizontalAlign = WebControls.HorizontalAlign.Center
   gvData.Width = 600
   gvData.PageSize = 5
   gvData.PagerSettings.Mode = PagerButtons.Numeric
   gvData.PagerSettings.PageButtonCount = 5
   gvData.PagerSettings.Position = PagerPosition.Bottom
   gvData.PagerStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
   gvData.PagerStyle.CssClass = "pagerText"

   gvData.HeaderStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
   gvData.HeaderStyle.CssClass = "tableHeader"
   gvData.RowStyle.CssClass = "tableCellNormal"
   gvData.AlternatingRowStyle.CssClass = "tableCellAlternating"

   'create the columns in the GridView
   gridColumn = New BoundField()
   gridColumn.DataField = "Title"
   gridColumn.HeaderText = "Title"
   gridColumn.SortExpression = "Title"
   gvData.Columns.Add(gridColumn)

   gridColumn = New BoundField()
   gridColumn.DataField = "PublishDate"
   gridColumn.HeaderText = "Publish Date"
   gridColumn.SortExpression = "PublishDate"
   gridColumn.DataFormatString = "{0:MMM dd, yyyy}"
   gridColumn.ItemStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
   gvData.Columns.Add(gridColumn)

   gridColumn = New BoundField()
   gridColumn.DataField = "ListPrice"
   gridColumn.HeaderText = "List Price"
   gridColumn.SortExpression = "ListPrice"
   gridColumn.DataFormatString = "{0:C2}"
   gridColumn.ItemStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
   gvData.Columns.Add(gridColumn)

   'add the event handler for the GridView RowCreated event
   AddHandler gvData.RowCreated, AddressOf gvData_RowCreated

   'add the GridView to the custom web part controls
   Me.Controls.Add(gvData)

   'create a SQL data source and set its properties to get the book data
   'from a database
   dSource = New SqlDataSource()
   dSource.ConnectionString = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
   dSource.DataSourceMode = SqlDataSourceMode.DataSet
```

```
 dSource.ProviderName = "System.Data.OleDb"
 dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " & _
    "FROM Book " & _
    "ORDER BY Title"
 dSource.ID = "dSource"

 'set the data source ID for the GridView
 'NOTE: The DataSourceID must be used instead of the DataSource if the
 '    automatic sorting/paging in GridView are to be used.
 gvData.DataSourceID = dSource.ID

 'add the SQL data source to the custom web part controls
 Me.Controls.Add(dSource)

 'perform the initial sort on the first column in ascending order
 'if no sort expression is currently set (this is the case when the
 'control is initially created)
 If (gvData.SortExpression.Length = 0) Then
  gvData.Sort(gvData.Columns(0).SortExpression, _
   SortDirection.Ascending)

 End If
End Sub 'CreateChildControls

 '''*************************************************************************
''' <summary>
''' This routine provides the event handler for the GridView's row created
''' event. It is responsible for setting the icon in the header row to
''' indicate the current sort column and sort order
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub gvData_RowCreated(ByVal sender As Object, _
  ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs)
 Dim index As Integer
 Dim col As DataControlField = Nothing
 Dim image As HtmlImage = Nothing

 If (e.Row.RowType = DataControlRowType.Header) Then
  'loop through the columns in the gridview updating the header to
  'mark which column is the sort column and the sort order
  For index = 0 To gvData.Columns.Count - 1
  col = gvData.Columns(index)

  'check to see if this is the sort column
  If (col.SortExpression.Equals(gvData.SortExpression)) Then
  'this is the sort column so determine whether the ascending or
  'descending image needs to be included
  image = New HtmlImage()
  image.Border = 0
  If (gvData.SortDirection = SortDirection.Ascending) Then
```

```
    image.Src = "images/sort_ascending.gif"
    Else
    image.Src = "images/sort_descending.gif"
    End If

    'add the image to the column header
    e.Row.Cells(index).Controls.Add(image)
    End If 'If (col.SortExpression = sortExpression)
    Next index
   End If 'If (gvData.SortExpression.Equals(String.Empty))
  End Sub 'gvData_RowCreated
 End Class 'CH11CustomWebPartVB
End Namespace
```

Example 11-13. Custom web part control (.cs)

```
using System;
using System.Drawing;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides a custom web part that displays book data
 /// </summary> public class CH11CustomWebPartCS : WebPart
 {

  private GridView gvData = null;

  ///*************************************************************************
  /// <summary>
  /// This routine creates the child controls (GridView and SqlDataSource)
  /// for the web part
  /// </summary>
  protected override void CreateChildControls()
  {
   BoundField gridColumn;
   SqlDataSource dSource;

   // clear the controls collection
   Controls.Clear();
```

```
// 'create the GridView control and set the applicable properties
// to display the book data
gvData = new GridView();
gvData.AllowPaging = true;
gvData.AllowSorting = true;
gvData.AutoGenerateColumns = false;
gvData.BorderColor = ColorTranslator.FromHtml("#000080");
gvData.BorderStyle = BorderStyle.Solid;
gvData.BorderWidth = 2;
gvData.Caption = "";
gvData.HorizontalAlign = HorizontalAlign.Center;
gvData.Width = 600;
gvData.PageSize = 5;
gvData.PagerSettings.Mode = PagerButtons.Numeric;
gvData.PagerSettings.PageButtonCount = 5;
gvData.PagerSettings.Position = PagerPosition.Bottom;
gvData.PagerStyle.HorizontalAlign = HorizontalAlign.Center;
gvData.PagerStyle.CssClass = "pagerText";

gvData.HeaderStyle.HorizontalAlign = HorizontalAlign.Center;
gvData.HeaderStyle.CssClass = "tableHeader";
gvData.RowStyle.CssClass = "tableCellNormal";
gvData.AlternatingRowStyle.CssClass = "tableCellAlternating";
// create the columns in the GridView
gridColumn = new BoundField();
gridColumn.DataField = "Title";
gridColumn.HeaderText = "Title";
gridColumn.SortExpression = "Title";
gvData.Columns.Add(gridColumn);

gridColumn = new BoundField();
gridColumn.DataField = "PublishDate";
gridColumn.HeaderText = "Publish Date";
gridColumn.SortExpression = "PublishDate";
gridColumn.DataFormatString = "{0:MMM dd, yyyy}";
gridColumn.ItemStyle.HorizontalAlign = HorizontalAlign.Center;
gvData.Columns.Add(gridColumn);

gridColumn = new BoundField();
gridColumn.DataField = "ListPrice";
gridColumn.HeaderText = "List Price";
gridColumn.SortExpression = "ListPrice";
gridColumn.DataFormatString = "{0:C2}";
gridColumn.ItemStyle.HorizontalAlign = HorizontalAlign.Center;
gvData.Columns.Add(gridColumn);

// add the event handler for the GridView RowCreated event
gvData.RowCreated += new GridViewRowEventHandler(gvData_RowCreated);

// add the GridView to the custom web part controls
this.Controls.Add(gvData);
```

```
    // create a SQL data source and set its properties to get the book data
    // from a database
    dSource = new SqlDataSource();
    dSource.ConnectionString = ConfigurationManager.
     ConnectionStrings["dbConnectionString"].ConnectionString;
    dSource.DataSourceMode = SqlDataSourceMode.DataSet;
    dSource.ProviderName = "System.Data.OleDb";
    dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice " +
       "FROM Book " +
       "ORDER BY Title";
    dSource.ID = "dSource";

    // set the data source ID for the GridView
    // NOTE: The DataSourceID must be used instead of the DataSource if the
    //   automatic sorting/paging in GridView are to be used.
    gvData.DataSourceID = dSource.ID;

    // add the SQL data source to the custom web part controls
    this.Controls.Add(dSource);

    // perform the initial sort on the first column in ascending order
    // if no sort expression is currently set (this is the case when the
    // control is initially created)
    if (gvData.SortExpression.Length == 0)
    {
     gvData.Sort(gvData.Columns[0].SortExpression,
     SortDirection.Ascending);
    }
   } // CreateChildControls

   ///***********************************************************************
   /// <summary>
   /// This routine provides the event handler for the GridView's row created
   /// event. It is responsible for setting the icon in the header row to
   /// indicate the current sort column and sort order
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void gvData_RowCreated(Object sender,
       System.Web.UI.WebControls.GridViewRowEventArgs e)
   {
    DataControlField col = null;
    HtmlImage image = null;

    if (e.Row.RowType == DataControlRowType.Header)
    {
     // loop through the columns in the gridview updating the header to
     // mark which column is the sort column and the sort order
     for (int index = 0; index < gvData.Columns.Count; index++)
     {
     col = gvData.Columns[index];
     // check to see if this is the sort column
     if (col.SortExpression.Equals(gvData.SortExpression))
     {
     // this is the sort column so determine whether the ascending or
     // descending image needs to be included
     image = new HtmlImage();
     image.Border = 0;
     if (gvData.SortDirection == SortDirection.Ascending)
     {
     image.Src = "images/sort_ascending.gif";
     }
     else
     {
     image.Src = "images/sort_descending.gif";
     }

     // add the image to the column header
     e.Row.Cells[index].Controls.Add(image);
     } // if (col.SortExpression.Equals(gvBooks.SortExpression))
     } // for index
    } // if (e.Row.RowType == DataControlRowType.Header)
   } //gvData_RowCreated
  } // CH11CustomWebPartCS
 }
```

Example 11-14. Using the custom web part (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH11UsingCustomWebPartVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH11UsingCustomWebPartVB"
 Title="Using a Custom Web Part" %>
<%@ Register TagPrefix="ASPNetCookbook" TagName="CvilleWeather"
     Src="~/CH11CVilleWeatherVB.ascx" %>
<%@ Register TagPrefix="ASPNetCookbook"
    Namespace="ASPNetCookbook.VBExamples"
    Assembly="__Code" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Using a Custom Web Part (VB)
 </div>
 <asp:WebPartManager ID="wpm1" runat="server" Personalization-Enabled="true" />
 <table width="90%" align="center" border="1" cellpadding="4" cellspacing="0">
  <tr>
   <td align="right">
    <asp:LinkButton ID="btnCustomize" runat="server"
```

```
        Text="Customize"
        CssClass="labelText"
        OnClick="btnCustomize_Click" />&nbsp;&nbsp;
    <asp:LinkButton ID="btnReset" runat="server"
        Text="Reset"
        CssClass="labelText"
        OnClick="btnReset_Click" />
   </td>
  </tr>
  <tr>
   <td>
     <asp:WebPartZone ID="webPartZone1" runat="server"
        EmptyZoneText="No Content Selected"
        Height="10" HeaderText="Zone 1"
        LayoutOrientation="Horizontal"
        CssClass="largeLabelText"
        Padding="6" />
   </td>
  </tr>
  <tr>
   <td>
     <asp:WebPartZone ID="webPartZone2" runat="server"
        EmptyZoneText="No Content Selected"
        Height="10" HeaderText="Zone 2"
        LayoutOrientation="Horizontal"
        CssClass="largeLabelText"
        Padding="6" />
   </td>
  </tr>
  <tr>
   <td>
     <asp:CatalogZone ID="CatalogZone1" runat="server"
        EmptyZoneText="No Catalog Items"
        HeaderCloseVerb-Visible="false"
        CssClass="largeLabelText"
        Padding="6" >
    <ZoneTemplate>
     <asp:DeclarativeCatalogPart ID="dpc1" runat="server"
             Title="Available Parts" >
    <WebPartsTemplate>
     <ASPNetCookbook:CvilleWeather ID="ucCvilleWeather"
              runat="server"
              Title="Weather" />
      <asp:Calendar ID="cal1" runat="server"
          Title="Calendar" />
     <ASPNetCookbook:CH11 CustomWebPartVB ID="cwpBooks" runat="server"
                 Title="Book Data" />
    </WebPartsTemplate>
    </asp:DeclarativeCatalogPart>
    </ZoneTemplate>
    </asp:CatalogZone>
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 11-15. Using the custom web part code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH11UsingCustomWebPartVB.aspx
 ''' </summary>
 Partial  Class  CH11UsingCustomWebPartVB
  Inherits System.Web.UI.Page
  '''**********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the customize button
  ''' click event. It is responsible for placing the web part manager
  ''' in catalog mode.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnCustomize_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
    wpm1.DisplayMode  =  WebPartManager.CatalogDisplayMode
  End Sub 'btnCustomize_Click

  '''**********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the reset button
  ''' click event. It is responsible for resetting the web part manager
  ''' personalization data.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnReset_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)
    wpm1.Personalization.ResetPersonalizationState()
  End Sub 'btnReset_Click
 End  Class  'CH11UsingCustomWebPartVB
End  Namespace
```

Example 11-16. Using the custom web part code-behind (.cs)

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

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH11UsingRegularContolsAsWebPartsCS.aspx
 /// </summary>
 public partial class CH11UsingCustomWebPartCS : System.Web.UI.Page
 {
  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the customize button
  /// click event. It is responsible for placing the web part manager
  /// in catalog mode.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnCustomize_Click(Object sender,
         System.EventArgs e)
  {
    wpm1.DisplayMode = WebPartManager.CatalogDisplayMode;
  } // btnCustomize_Click

  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the reset button
  /// click event. It is responsible for resetting the web part manager
  /// personalization data.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnReset_Click(Object sender,
        System.EventArgs e)
  {
    wpm1.Personalization.ResetPersonalizationState();
  } // btnReset_Click
 } // CH11UsingCustomWebPartCS
}
```

PREV

PREV

Recipe 11.5. Communicating Between Web Parts

Problem

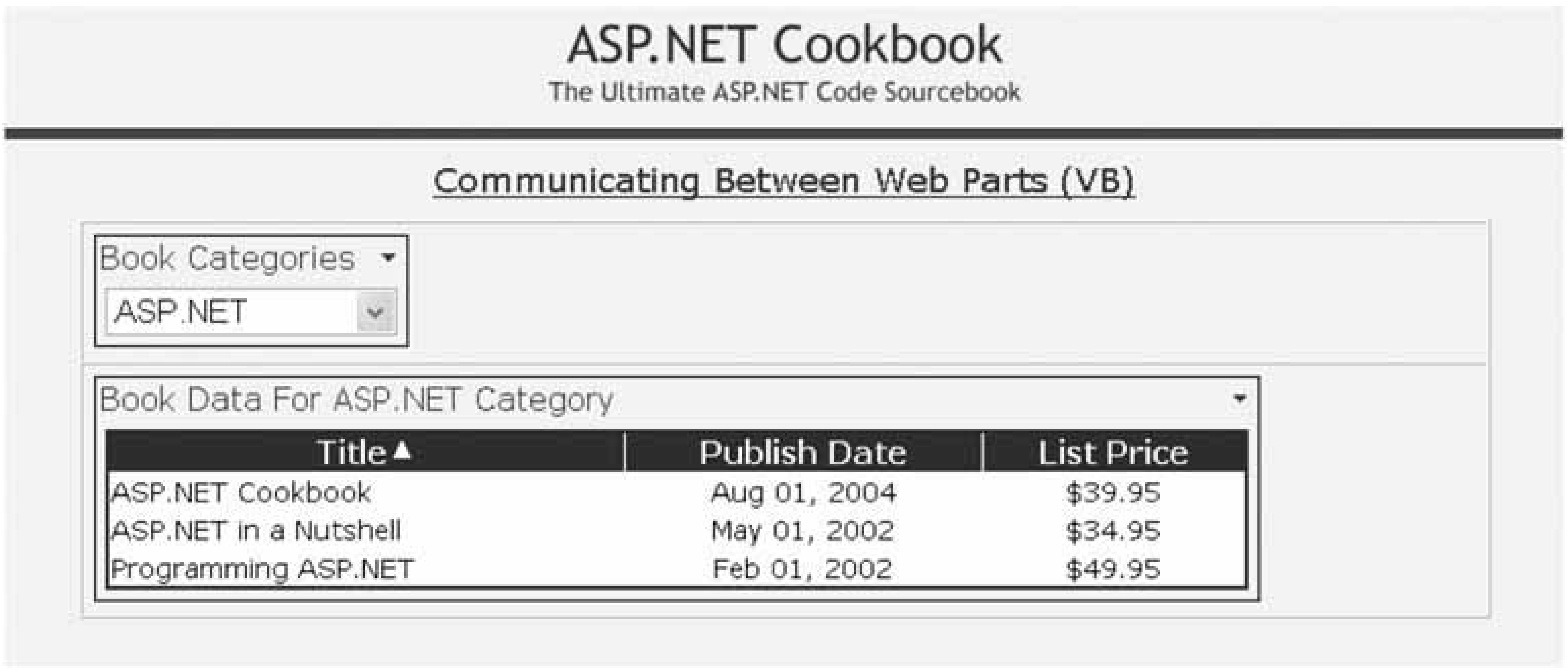
You have a web part in your application that needs to notify another web part when changes are made by the user. For example, suppose you have one web part that reads data from a database and acts a filter for a second web part displaying the filtered data. When the user changes the filter settings in the first web part, you want the second web part to be notified accordingly so it can display the new data.

Solution

Create a provider web part control, a consumer web part control, an interface to use as the communication contract between the web parts, and a page that uses both web part controls. In the provider web part control, add a method that returns the interface and is marked as the connection provider. In the consumer web part control, add a method that is passed the interface and is marked as the connection consumer. In the page that uses the web part controls, add a`WebPartConnection` control to the `<StaticConnections>` element of the `WebPartManager` defining the connection between the web parts.

The application we have implemented to demonstrate the solution is shown in Examples 11-17, 11-18 , 11-19 , 11-20 , 11-21 , 11-22 through 11-23 . Examples 11-17 (VB) and 11-18 (C#) show the interface used for messages between the provider and consumer web parts. Examples 11-19 (VB) and 11-20 (C#) show the provider web part. Examples 11-21 (VB) and 11-22 (C#) show the consumer web part. Example 11-23 shows the *.aspx* for a page that demonstrates communicating between web parts. No code-behind is shown for the demonstration page because no code is required. Figure 11-7 shows the result.

Figure 11-7. Output of page demonstrating the communication between web parts


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Communicating Between Web Parts (VB)

Book Categories
ASP.NET

Book Data For ASP.NET Category

Title ▲	Publish Date	List Price
ASP.NET Cookbook	Aug 01, 2004	$39.95
ASP.NET in a Nutshell	May 01, 2002	$34.95
Programming ASP.NET	Feb 01, 2002	$49.95

Discussion

The web parts framework provides the ability to communicate between web parts. The communication approach is a bit different from the scheme used to communicate between user controls (see Recipe 5.4). Instead of broadcasting an event, which can be consumed by many user controls, web part communication is explicitly "wired" between two web parts. Web parts can participate in communication with multiple web parts but each communication path consists of a single provider web part and a single consumer web part.

The communication path can be static (defined declaratively in the page that uses the controls) or dynamic (created programmatically or by the user). This recipe describes creating a static connectior between web parts. For more information on connections between web parts, refer to the documentation on the `WebPartConnection` class in the MSDN Library.

Establishing a static connection between web parts requires four steps:

1. Define an interface that will be used as the communication contract between the web parts.
2. Add a method to the provider web part that returns the interface and is decorated with the `ConnectionProvider` attribute.
3. Add a method to the consumer web part that is passed the interface and is decorated with the `ConnectionConsumer` attribute.
4. Add a `WebPartConnection` control in the `<StaticConnections>` element of the `WebPartManager` control in the page where the web parts are used specifying the ID of the provider web part anc the ID of the consumer web part.

Once these four steps have been implemented, the web part infrastructure handles calling the provider to get the data and then calling the consumer to supply the data.

In our application that implements this solution, our provider web part contains a`DropDownList` control with the categories of books in a database. Our consumer web part displays a list of books fo the selected category.

The interface we created as the communication contract defines a single property (`BookCategory`) that provides the ability to pass the selected book category from the provider web part to the consumer web part.

VB

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This interface provides the definition of the definition of the
 ''' properties shared between web parts
 ''' </summary>
 Public Interface IBookCategoryVB
  '''*********************************************************************
  ''' <summary>
  ''' This property defines the category for the book data
  ''' </summary>
  Property BookCategory() As String

 End Interface
End Namespace
```

```
using System;

/// <summary>
/// This interface provides the definition of the definition of the
/// properties shared between web parts
/// </summary>
namespace ASPNetCookbook.CSExamples
{
 public interface IBookCategoryCS
 {
  ///*********************************************************************
  /// <summary>
  /// This property defines the category for the book data
  /// </summary>
  String BookCategory
  {
   get;
   set;
  }
 } // IBookCategoryCS
}
```

The `CreateChildControls` method of the provider web part class creates a `DropDownList` and a `SqlDataSource` control to display the available book categories in the same manner as described in Recipe 11.3. In addition, the class has a constructor that disables the Close button in the web part tit bar since we do not want the user to be able to close the web part in this example.

VB

```
Me.AllowClose = False
```

C#

```
this.AllowClose = false;
```

The provider web part also has a `BookCategory` property to get the currently selected category from the `DropDownList` and to select an entry in the `DropDownList` when the property is being set:

VB

```
Public Property BookCategory( ) As String _
  Implements IBookCategoryVB.BookCategory
  Get
    'make sure the child controls have been created
    EnsureChildControls( )
    Return (ddBookCategories.SelectedValue)
  End Get

  Set(ByVal value As String)
    'make sure the child controls have been created
    EnsureChildControls( )

    'find the item matching the passed value in the drop down list
    'and select it
    ddBookCategories.SelectedIndex = _
      ddBookCategories.Items.IndexOf(ddBookCategories.Items.FindByValue(value))
  End Set
End Property
```

```
public String BookCategory
{
  get
  {
    // make sure the child controls have been created
    EnsureChildControls( );
    return (ddBookCategories.SelectedValue);
}

  set
  {
    // make sure the child controls have been created
    EnsureChildControls( );
    // find the item matching the passed value in the drop down list
    // and select it
    ddBookCategories.SelectedIndex =
     ddBookCategories.Items.IndexOf(ddBookCategories.Items.FindByValue(value));
   }
  }
```

The setter and getters of the property make a call to the `EnsureChildControls` method of the web part. This method checks to see if the child controls have been created; if they have not, it will make the necessary calls to create the child controls. If the call to `EnsureChildControls` is not made, it will be possible for the property to be accessed before the child controls are created, resulting in an exception being thrown because the `ddBookCategories` control is null.

The `GetConnectionInterface` method of the provider web part is the method called by the web part infrastructure to get the book category data. The method returns a reference to the instance of the provider web part. It is decorated with the `ConnectionProvider` attribute to define the method used as the provider of the data in the communications.

```
<ConnectionProvider("BookCategory")> _
Public Function GetConnectionInterface( ) As IBookCategoryVB
 Return Me
End Function 'GetConnectionInterface
```

```
[ConnectionProvider("BookCategory")]
public IBookCategoryCS GetConnectionInterface( )
{
 return (this);
} // GetConnectionInterface
```

The consumer web part in our example is a modification of the book list web part created in Recipe 11.3. We have added a constructor to disable the Close button in the web part title bar in the same manner as we did for the provider web part.

We have added a `BookCategory` property and a `SetConnectionInterface` method. The `BookCategory` property provides the ability to get and set the book category used to filter the list of books retrieved from a database.

The `SetConnectionInterface` method of the consumer web part is called by the web part infrastructure to pass the book category data read from the provider web part. It is decorated with the `ConnectionConsumer` attribute to define the method used as the consumer of the data in the communications.

The `SetConnectionInterface` method sets the `BookCategory` property, filters the data in the data source to include only the books in the selected category, and sets the title of the web part to include the book category:

VB

```
<ConnectionConsumer("BookCategory")> _
Public Sub SetConnectionInterface(ByVal bookCategoryInterface As IBookCategoryVB)

  'make the child controls have been created
  EnsureChildControls( )

  'set the book category to the selected category
  BookCategory = bookCategoryInterface.BookCategory
```

VB

```
  'filter the data source for the selected category
  dSource.FilterExpression = "Category='" &BookCategory &"'"

  'set the title of the web part to include the selected category
  MyBase.Title = "Book Data For " &BookCategory &" Category"
End Sub 'SetConnectionInterface
```

```
[ConnectionConsumer("BookCategory")]
public void SetConnectionInterface(IBookCategoryCS bookCategoryInterface)
{
  // make the child controls have been created
  EnsureChildControls( );

  // set the book category to the selected category
  BookCategory = bookCategoryInterface.BookCategory;

  // filter the data source for the selected category
  dSource.FilterExpression = "Category='" + BookCategory + "'";

  // set the title of the web part to include the selected category
  base.Title = "Book Data For " + BookCategory + " Category";
} // SetConnectionInterface
```

The final step to making it all work is to define the connection between the provider and consumer web parts. We do this by adding `WebPartConnection` control to the `<StaticConnections>` element of the `WebPartManager` in the demonstration page, setting the `ProviderID` attribute to the ID of the provider web part and the `ConsumerID` attribute to the ID of the consumer web part.

```
<asp:WebPartManager  ID="wpm1"  runat="server"  >
  <StaticConnections>
    <asp:WebPartConnection ID="wpc1"
           ConsumerID="cwpBooks"
```

```
        ProviderID="cwpCategories" />
  </StaticConnections>
 </asp:WebPartManager>
```

Adding communications between web parts improves their reusability. By creating task-focused web parts, such as providing the ability for the user to select a book category in one web part and displaying the data in another web part, the controls can be reused for different tasks in other pages and applications.

Web part communication is not limited to custom web parts. Custom server controls and user controls can use the same techniques described in this recipe to participate in communications between web parts.

See Also

Recipe 5.4 and 11.3

Example 11-17. Interface class used as the message between web parts (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This interface provides the definition of the definition of the
 ''' properties shared between web parts
 ''' </summary>
 Public Interface IBookCategoryVB
  '''*****************************************************************************
  ''' <summary>
  ''' This property defines the category for the book data
  ''' </summary>
  Property BookCategory( ) As String

 End Interface
End Namespace
```

Example 11-18. Interface class used as the message between web parts (.cs)

```
using System;

/// <summary>
/// This interface provides the definition of the definition of the
/// properties shared between web parts
/// </summary>
namespace ASPNetCookbook.CSExamples
{
 public interface IBookCategoryCS
 {
  ///*********************************************************************
  /// <summary>
  /// This property defines the category for the book data
  /// </summary>
  String BookCategory
  {
   get;
   set;
  }
 } // IBookCategoryCS
}
```

Example 11-19. Communicating between web partsprovider web part control (.vb)

```
Option Explicit On
Option Strict On

Imports System.Configuration
Imports System.Data
Imports System.Data.SqlClient

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides a custom web part that displays the available
 ''' book categories and provides the ability to notify other web parts
 ''' when the user selects a new category
 ''' </summary>
 Public Class CH11BookCategoryWebPartVB
  Inherits WebPart
  Implements IBookCategoryVB

  Private ddBookCategories As DropDownList

  '''*********************************************************************
  ''' <summary>
  ''' This property provides the ability to get the selected book category
  ''' </summary>
  Public Property BookCategory( ) As String _
   Implements IBookCategoryVB.BookCategory
   Get
    'make sure the child controls have been created
    EnsureChildControls( )
    Return (ddBookCategories.SelectedValue)
   End Get

   Set(ByVal value As String)
    'make sure the child controls have been created
    EnsureChildControls( )

    'find the item matching the passed value in the drop down list
    'and select it
    ddBookCategories.SelectedIndex = _
    ddBookCategories.Items.IndexOf(ddBookCategories.Items.FindByValue(value))
   End Set
  End Property

  '''**************************************************************************
  ''' <summary>
  ''' This routine provides the interface for obtaining the selected
  ''' category when the data has changed
  ''' </summary>
  <ConnectionProvider("BookCategory")> _
  Public Function GetConnectionInterface( ) As IBookCategoryVB
   Return Me
  End Function 'GetConnectionInterface

  '''**************************************************************************
  ''' <summary>
  ''' This constructor initializes the web part to default values
  ''' </summary>
  ''' <remarks></remarks>
  Public Sub New( )
   'disable the "close" verb
   Me.AllowClose = False
  End Sub 'New
  '''**************************************************************************
  ''' <summary>
  ''' This routine creates the child controls (book category DropDownList)
  ''' for the web part
  ''' </summary>
  Protected Overrides Sub CreateChildControls( )
   Dim dSource As SqlDataSource

   'clear the controls collection
   Controls.Clear( )

   'create the dropdown list and add it to the custom web part controls
```

```
   ddBookCategories = New DropDownList( )
   ddBookCategories.AutoPostBack = True
   Me.Controls.Add(ddBookCategories)

   'create a SQL data source and set its properties to get the book
   'categories from a database
   dSource = New SqlDataSource( )
   dSource.ConnectionString = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
   dSource.DataSourceMode = SqlDataSourceMode.DataSet
   dSource.ProviderName = "System.Data.OleDb"
   dSource.SelectCommand = "SELECT DISTINCT Category " &_
      "FROM Book " &_
      "ORDER BY Category ASC"


   'set the data source for the DropDownList and bind the data
   ddBookCategories.DataSource = dSource
   ddBookCategories.DataTextField = "Category"
   ddBookCategories.DataBind( )

   'add the SQL data source to the custom web part controls
   Me.Controls.Add(dSource)
  End Sub 'CreateChildControls
 End Class 'CH11BookCategoryWebPartVB
End Namespace
```

Example 11-20. Communicating between web partsprovider web part control (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides a custom web part that displays book data
 /// </summary>
 public class CH11BookCategoryWebPartCS :
  WebPart, IBookCategoryCS
 {

  private DropDownList ddBookCategories = null;
```

```
///***************************************************************************
/// <summary>
/// This property provides the ability to get the selected book category
/// </summary>
public String BookCategory
{
 get
 {
  // make sure the child controls have been created
  EnsureChildControls( );
  return (ddBookCategories.SelectedValue);
}
 set
 {
  // make sure the child controls have been created
  EnsureChildControls( );

  // find the item matching the passed value in the drop down list
  // and select it
  ddBookCategories.SelectedIndex =
  ddBookCategories.Items.IndexOf(ddBookCategories.Items.FindByValue(value));
 }
}

///***************************************************************************
/// <summary>
/// This routine provides the interface for obtaining the selected
/// category when the data has changed
/// </summary>
[ConnectionProvider("BookCategory")]
public IBookCategoryCS GetConnectionInterface( )
{
 return (this);
} // GetConnectionInterface

///***************************************************************************
/// <summary>
/// This constructor initializes the web part to default values
/// </summary>
/// <remarks></remarks>
public CH11BookCategoryWebPartCS( )
{
 // disable the "close" verb
 this.AllowClose = false;
} // CH11BookCategoryWebPartCS

///***************************************************************************
/// <summary>
/// This routine creates the child controls (book category DropDownList)
/// for the web part
/// </summary>
```

```
  protected override void CreateChildControls( )
  {
   SqlDataSource dSource;

   // clear the controls collection
   Controls.Clear( );

   // create the dropdown list and add it to the custom web part controls
   ddBookCategories = new DropDownList( );
   ddBookCategories.AutoPostBack = true;
   this.Controls.Add(ddBookCategories);

   // create a SQL data source and set its properties to get the book
   // categories from a database
   dSource = new SqlDataSource( );
   dSource.ConnectionString = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
   dSource.DataSourceMode = SqlDataSourceMode.DataSet;
   dSource.ProviderName = "System.Data.OleDb";
   dSource.SelectCommand = "SELECT DISTINCT Category " +
      "FROM Book " +
      "ORDER BY Category ASC";

   // set the data source for the DropDownList and bind the data
   ddBookCategories.DataSource = dSource;
   ddBookCategories.DataTextField = "Category";
   ddBookCategories.DataBind( );

   // add the SQL data source to the custom web part controls
   this.Controls.Add(dSource);
  } // CreateChildControls
 } //  CH11BookCategoryWebPartCS
}
```

Example 11-21. Communicating between web partsconsumer web part control (.vb)

```
Option Explicit On
Option Strict On

Imports System.Configuration
Imports System.Data
Imports System.Data.SqlClient

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides a custom web part that displays book data
 ''' and provides the ability to be notified by another web part
 ''' that the selected book category has changed
 ''' </summary>
 Public  Class  CH11CustomWebPartWithCommVB
  Inherits WebPart

  Private gvData As GridView
  Private dSource As SqlDataSource
  Private mBookCategory As String

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the BookCategory
  ''' </summary>
  Public Property BookCategory( ) As String
   Get
    Return (mBookCategory)
   End Get

   Set(ByVal value As String)
    mBookCategory = value
   End Set
  End Property

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the interface that is called to be notified
  ''' that the selected category has been changed
  ''' </summary>
  '''
  ''' <param name="bookCategoryInterface">Set to book category data</param>
  <ConnectionConsumer("BookCategory")> _
  Public Sub SetConnectionInterface( _
      ByVal bookCategoryInterface As IBookCategoryVB)

   'make the child controls have been created
   EnsureChildControls( )

   'set the book category to the selected category
   BookCategory = bookCategoryInterface.BookCategory

   'filter the data source for the selected category
   dSource.FilterExpression = "Category='" &BookCategory &"'"

   'set the title of the web part to include the selected category
   MyBase.Title = "Book Data For " &BookCategory &" Category"
  End Sub 'SetConnectionInterface

  '''*************************************************************************
  ''' <summary>
  ''' This constructor initializes the web part to default values
  ''' </summary>
```

```
Public Sub New( )
 'set the default title
 MyBase.Title = "Book Data For All Categories"

 'disable the "close" verb
 Me.AllowClose = False
End Sub 'New

'''**************************************************************************
''' <summary>
''' This routine creates the child controls (GridView and SqlDataSource)
''' for the web part
''' </summary>
Protected Overrides Sub CreateChildControls( )
 Dim gridColumn As BoundField

 'clear the controls collection
 Controls.Clear( )

 'create the GridView control and set the applicable properties
 'to display the book data
 gvData = New GridView( )
 gvData.AllowPaging = True
 gvData.AllowSorting = True
 gvData.AutoGenerateColumns = False
 gvData.BorderColor = Drawing.ColorTranslator.FromHtml("#000080")
 gvData.BorderStyle = WebControls.BorderStyle.Solid
 gvData.BorderWidth = 2
 gvData.Caption = ""
 gvData.HorizontalAlign = WebControls.HorizontalAlign.Center
 gvData.Width = 600
 gvData.PageSize = 5
 gvData.PagerSettings.Mode = PagerButtons.Numeric
 gvData.PagerSettings.PageButtonCount = 5
 gvData.PagerSettings.Position = PagerPosition.Bottom
 gvData.PagerStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
 gvData.PagerStyle.CssClass = "pagerText"

 gvData.HeaderStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
 gvData.HeaderStyle.CssClass = "tableHeader"
 gvData.RowStyle.CssClass = "tableCellNormal"
 gvData.AlternatingRowStyle.CssClass = "tableCellAlternating"

 'create the columns in the GridView
 gridColumn = New BoundField( )
 gridColumn.DataField = "Title"
 gridColumn.HeaderText = "Title"
 gridColumn.SortExpression = "Title"
 gvData.Columns.Add(gridColumn)

 gridColumn = New BoundField( )
 gridColumn.DataField = "PublishDate"
 gridColumn.HeaderText = "Publish Date"

 gridColumn.SortExpression = "PublishDate"
  gridColumn.DataFormatString = "{0:MMM dd, yyyy}"
 gridColumn.ItemStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
 gvData.Columns.Add(gridColumn)

 gridColumn = New BoundField( )
 gridColumn.DataField = "ListPrice"
 gridColumn.HeaderText = "List Price"
 gridColumn.SortExpression = "ListPrice"
  gridColumn.DataFormatString = "{0:C2}"
 gridColumn.ItemStyle.HorizontalAlign = WebControls.HorizontalAlign.Center
 gvData.Columns.Add(gridColumn)

 'add the event handler for the GridView RowCreated event
 AddHandler gvData.RowCreated, AddressOf gvData_RowCreated

 'add the GridView to the custom web part controls
 Me.Controls.Add(gvData)

 'create a SQL data source and set its properties to get the book data
 'from a database
 dSource = New SqlDataSource( )
 dSource.ConnectionString = ConfigurationManager. _
  ConnectionStrings("dbConnectionString").ConnectionString
 dSource.DataSourceMode = SqlDataSourceMode.DataSet
 dSource.ProviderName = "System.Data.OleDb"
 dSource.ID = "dSource"
 dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice, Category " &_
    "FROM Book " &_
    "ORDER BY Title"

 'set the data source ID for the GridView
 'NOTE: The DataSourceID must be used instead of the DataSource if the
 '    automatic sorting/paging in GridView are to be used.
 gvData.DataSourceID = dSource.ID

 'add the SQL data source to the custom web part controls
 Me.Controls.Add(dSource)

 'perform the initial sort on the first column in ascending order
 'if no sort expression is currently set (this is the case when the
 'control is initially created)
  If (gvData.SortExpression.Length = 0) Then
    gvData.Sort(gvData.Columns(0).SortExpression, _
  SortDirection.Ascending)
 End If
End Sub 'CreateChildControls

 '''**************************************************************************
''' <summary>
```

```
  ''' This routine provides the event handler for the GridView's row created
  ''' event. It is responsible for setting the icon in the header row to
  ''' indicate the current sort column and sort order
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub gvData_RowCreated(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs)
   Dim index As Integer
   Dim col As DataControlField = Nothing
   Dim image As HtmlImage = Nothing

   If (e.Row.RowType = DataControlRowType.Header) Then
    'loop through the columns in the gridview updating the header to
    'mark which column is the sort column and the sort order
    For index = 0 To gvData.Columns.Count - 1
    col = gvData.Columns(index)

    'check to see if this is the sort column
    If (col.SortExpression.Equals(gvData.SortExpression)) Then
    'this is the sort column so determine whether the ascending or
    'descending image needs to be included
    image = New HtmlImage( )
    image.Border = 0
    If (gvData.SortDirection = SortDirection.Ascending) Then
    image.Src = "images/sort_ascending.gif"
    Else
    image.Src = "images/sort_descending.gif"
    End If

    'add the image to the column header
    e.Row.Cells(index).Controls.Add(image)
    End If 'If (col.SortExpression = sortExpression)
    Next index
   End If 'If (gvData.SortExpression.Equals(String.Empty))
  End Sub
 End  Class  'CH11CustomWebPartWithCommVB
End Namespace
```

Example 11-22. Communicating between web partsconsumer web part control (.cs)

```
using System;
using System.Drawing;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
    /// This class provides a custom web part that displays book data
 /// and provides the ability to be notified by another web part
 /// that the selected book category has changed
 /// </summary>
 public class CH11CustomWebPartWithCommCS : WebPart
 {

  private GridView gvData = null;
  private SqlDataSource dSource = null;
  private String mBookCategory = null;

  ///**************************************************************************
  /// <summary>
  /// This property provides the ability to get/set the BookCategory
  /// </summary>
  public String BookCategory
  {
   get
   {
    return (mBookCategory);
   }

   set
   {
    mBookCategory = value;
   }
  }

  ///**************************************************************************
  /// <summary>
  /// This routine provides the interface that is called to be notified
  /// that the selected category has been changed
  /// </summary>
  ///
  /// <param name="bookCategoryInterface">Set to book category data</param>
  [ConnectionConsumer("BookCategory")]
  public void SetConnectionInterface(IBookCategoryCS bookCategoryInterface)
  {
   // make the child controls have been created
   EnsureChildControls( );

   // set the book category to the selected category
   BookCategory = bookCategoryInterface.BookCategory;
   // filter the data source for the selected category
   dSource.FilterExpression = "Category='" + BookCategory + "'";

   // set the title of the web part to include the selected category
   base.Title = "Book Data For " + BookCategory + " Category";
  } // SetConnectionInterface
  ///***************************************************************************
  /// <summary>
  /// This constructor initializes the web part to default values
  /// </summary>
  /// <remarks></remarks>
  public CH11CustomWebPartWithCommCS( )
  {
   /// set the default title
   base.Title = "Book Data For All Categories";

   // disable the "close" verb
   this.AllowClose = false;
  } // CH11CustomWebPartWithCommCS

  ///***************************************************************************
  /// <summary>
  /// This routine creates the child controls (GridView and SqlDataSource)
  /// for the web part
  /// </summary>
  protected override void CreateChildControls( )
  {
   BoundField gridColumn;

   // clear the controls collection
   Controls.Clear( );

   // create the GridView control and set the applicable properties
   // to display the book data
   gvData = new GridView( );
   gvData.AllowPaging = true;
   gvData.AllowSorting = true;
   gvData.AutoGenerateColumns = false;
    gvData.BorderColor = ColorTranslator.FromHtml("#000080");
   gvData.BorderStyle = BorderStyle.Solid;
   gvData.BorderWidth = 2;
   gvData.Caption = "";
   gvData.HorizontalAlign = HorizontalAlign.Center;
   gvData.Width = 600;
   gvData.PageSize = 5;
   gvData.PagerSettings.Mode = PagerButtons.Numeric;
   gvData.PagerSettings.PageButtonCount = 5;
   gvData.PagerSettings.Position = PagerPosition.Bottom;
   gvData.PagerStyle.HorizontalAlign = HorizontalAlign.Center;
   gvData.PagerStyle.CssClass = "pagerText";

   gvData.HeaderStyle.HorizontalAlign = HorizontalAlign.Center;
   gvData.HeaderStyle.CssClass = "tableHeader";
   gvData.RowStyle.CssClass = "tableCellNormal";
   gvData.AlternatingRowStyle.CssClass = "tableCellAlternating";

   // create the columns in the GridView
   gridColumn = new BoundField( );
   gridColumn.DataField = "Title";
   gridColumn.HeaderText = "Title";
   gridColumn.SortExpression = "Title";
   gvData.Columns.Add(gridColumn);

   gridColumn = new BoundField( );
   gridColumn.DataField = "PublishDate";
   gridColumn.HeaderText = "Publish Date";
   gridColumn.SortExpression = "PublishDate";
   gridColumn.DataFormatString = "{0:MMM dd, yyyy}";
   gridColumn.ItemStyle.HorizontalAlign = HorizontalAlign.Center;
   gvData.Columns.Add(gridColumn);

   gridColumn = new BoundField( );
   gridColumn.DataField = "ListPrice";
   gridColumn.HeaderText = "List Price";
   gridColumn.SortExpression = "ListPrice";
   gridColumn.DataFormatString = "{0:C2}";
   gridColumn.ItemStyle.HorizontalAlign = HorizontalAlign.Center;
   gvData.Columns.Add(gridColumn);

   // add the event handler for the GridView RowCreated event
   gvData.RowCreated += new GridViewRowEventHandler(gvData_RowCreated);

   // add the GridView to the custom web part controls
   this.Controls.Add(gvData);

   // create a SQL data source and set its properties to get the book data
   // from a database
   dSource = new SqlDataSource( );
   dSource.ConnectionString = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
   dSource.DataSourceMode = SqlDataSourceMode.DataSet;
   dSource.ProviderName = "System.Data.OleDb";
   dSource.ID = "dSource";
   dSource.SelectCommand = "SELECT Title, PublishDate, ListPrice, Category " +
      "FROM Book " +
      "ORDER BY Title";

   // set the data source ID for the GridView
   // NOTE: The DataSourceID must be used instead of the DataSource if the
   //   automatic sorting/paging in GridView are to be used.
   gvData.DataSourceID = dSource.ID;

   // add the SQL data source to the custom web part controls
   this.Controls.Add(dSource);
```

```
    // perform the initial sort on the first column in ascending order
    // if no sort expression is currently set (this is the case when the
    // control is initially created)
     if (gvData.SortExpression.Length == 0)
    {
      gvData.Sort(gvData.Columns[0].SortExpression,
     SortDirection.Ascending);
    }
  } // CreateChildControls

   ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the GridView's row created
  /// event. It is responsible for setting the icon in the header row to
  /// indicate the current sort column and sort order
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void gvData_RowCreated(Object sender,
    System.Web.UI.WebControls.GridViewRowEventArgs e)
  {
   DataControlField col = null;
   HtmlImage image = null;

   if (e.Row.RowType == DataControlRowType.Header)
   {
    // loop through the columns in the gridview updating the header to
    // mark which column is the sort column and the sort order
     for (int index = 0; index < gvData.Columns.Count; index++)
    {
    col = gvData.Columns[index];

    // check to see if this is the sort column
    if (col.SortExpression.Equals(gvData.SortExpression))
    {
    // this is the sort column so determine whether the ascending or
    // descending image needs to be included
    image = new HtmlImage( );
    image.Border = 0;
    if (gvData.SortDirection == SortDirection.Ascending)
    {
    image.Src = "images/sort_ascending.gif";
    }
    else
    {
    image.Src = "images/sort_descending.gif";
    }

    // add the image to the column header
    e.Row.Cells[index].Controls.Add(image);
    } // if (col.SortExpression.Equals(gvBooks.SortExpression))
    } // for index
   } // if (e.Row.RowType == DataControlRowType.Header)
  } //gvData_RowCreated
 } //  CH11CustomWebPartWithCommCS
}
```

Example 11-23. Communicating between web partsdemonstration page (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH11TestCommunicatingBetweenWebPartsVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH11TestCommunicatingBetweenWebPartsVB"
 Title="Communicating Between Web Parts" %>
<%@ Register TagPrefix="ASPNetCookbook"
    Namespace="ASPNetCookbook.VBExamples"
    Assembly="_ _Code" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Communicating Between Web Parts (VB)
 </div>
 <asp:WebPartManager ID="wpm1" runat="server" >
  <StaticConnections>
   <asp:WebPartConnection ID="wpc1"
        ConsumerID="cwpBooks"
        ProviderID="cwpCategories" />
  </StaticConnections>
 </asp:WebPartManager>
 <table width="90%" align="center" border="1" cellpadding="4" cellspacing="0">
  <tr>
   <td>
    <asp:WebPartZone ID="webPartZone1" runat="server"
        CssClass="largeLabelText">
    <ZoneTemplate>
    <ASPNetCookbook:CH11BookCategoryWebPartVB ID="cwpCategories"
    runat="server"
    Title="Book Categories" />
    </ZoneTemplate>
    </asp:WebPartZone>
    </td>
   </tr>
  <tr>
   <td>
    <asp:WebPartZone ID="webPartZone2" runat="server"
        CssClass="largeLabelText">
    <ZoneTemplate>
    <ASPNetCookbook:CH11CustomWebPartWithCommVB ID="cwpBooks"
    runat="server"
    Title="Book Data" />
    </ZoneTemplate>
    </asp:WebPartZone>
   </td>
  </tr>
 </table>
</asp:Content>
```

PREV

Recipe 11.6. Persisting Personalized Web Part Properties

Problem

You have created a web part with custom properties and you want the property data to be persisted along with the other web part personalization data so the next time the user revisits the page, his property settings are present.

Solution

Decorate the properties in your web part that you want persisted with the`Personalizable` attribute:

```
<Personalizable( )> _
Public Property BookCategory( ) As String
 Get

  …

 End Get

 Set(ByVal value As String)

  …

 End Set
End Property
```

```
[Personalizable( )]
public String BookCategory
{
 get
 {
  …
 }

 set
 {
  …
 }
}
```

Discussion

The web part infrastructure automatically handles persisting the personalization performed by the user when he adds web parts to pages. Which web parts the user selected and their location on the page is automatically stored and retrieved when the user revisits your site. You do not have to write any code to make this happen.

Adding custom property data to the persisted data is straightforward and requires only a small modification to your code. You will need to add the `Personalizable` attribute to each of the properties you want persisted. No other modifications are required.

To demonstrate this technique, we added the `Personalizable` attribute to the `BookCategory` property of the book category web part described in Recipe 11.4, as shown below. With this addition, the book category the user last selected is preselected for her on subsequent requests for the page.

VB

```
<Personalizable( )> _
Public Property BookCategory( ) As String _
 Implements IBookCategoryVB.BookCategory
 Get
  'make sure the child controls have been created
  EnsureChildControls( )
  Return (ddBookCategories.SelectedValue)
 End Get

 Set(ByVal value As String)
  'make sure the child controls have been created
  EnsureChildControls( )

  'find the item matching the passed value in the drop down list
  'and select it
  ddBookCategories.SelectedIndex = _
   ddBookCategories.Items.IndexOf(ddBookCategories.Items.FindByValue(value))
 End Set
End Property


[Personalizable( )]
public String BookCategory
{
 get
 {
  // make sure the child controls have been created
  EnsureChildControls( );
  return (ddBookCategories.SelectedValue);
 }

 set
```

```
   {
    // make sure the child controls have been created
    EnsureChildControls( );

    // find the item matching the passed value in the drop down list
    // and select it
    ddBookCategories.SelectedIndex =
      ddBookCategories.Items.IndexOf(ddBookCategories.Items.FindByValue(value));
   }
  }
```

Storing custom property data for your web parts is a great way to enhance the user experience for your application.

See Also

Recipe 11.4

PREV

Chapter 12. Configuration

PREV

12.0 Introduction

ASP.NET provides a convenient, extensible, XML-based mechanism for configuring ASP.NET and applications that run under it. It is an improvement over the IIS metabase that was awkward to change, required IIS to be restarted, and was not easily replicated on additional servers. By contrast ASP.NET automatically detects changes to the *web.config* file and transparently restarts the application; there is no need to restart IIS. Replicating an ASP.NET application configuration is as simple as copying the *web.config* file to the new server.

Configuration File Hierarchy

ASP.NET uses a hierarchy of configuration files. The *machine.config* file contains the settings for the server and is located in the *%SystemRoot%\Microsoft.NET\Framework\%VersionNumber%\CONFIG\* folder. You can create *web.config* files to configure each of your applications, overriding the settings in *machine.config*. Folders within the application can have *web.config* files to customize the configuration of portions of the application and override the settings in the *.config* files higher up the hierarchy.

Structure and Use of web.config

The basic structure of *web.config* has a format similar to *machine.config*. The idea is that you add to *web.config* only those entries for which you want to override settings in *machine.config*. At a minimum, *web.config* must have a `<configuration>` element and a child element, such as a `<system.web>` element. The following is a minimal *web.config* file:

```
<?xml  version="1.0"?>
<configuration>
 <system.web>
 </system.web>
   </configuration>
```

The `<configuration>` and `<system.web>` elements don't do anything special other than provide the structure for other settings you want to add. By using child elements of these default configuration elements in *web.config*, you can, for example, change the ASP.NET HTTP runtime settings, store `key/value` pairs in the `<appSettings>` element of *web.config*, and add elements of your own to *web.config*. The recipes in this chapter will show you how to do all these things and more.

Modifying web.config

The *web.config* file is an XML file and can be edited with any text editor. As with most XML, the structure of the *web.config* file must conform to a defined schema. (In this case, the schema is detailed in the Microsoft documentation.) The *web.config* must be a properly formed XML document. In other words, elements must be placed in the correct sections of the XML document and conform tc the case style and exact spelling defined in the schema. If a *web.config* file deviates from the schema in any detail, ASP.NET will throw a configuration error exception.

This chapter does not attempt to address all of the configuration settings available in *machine.config* and *web.config*. Rather, it provides information on many commonly used features and several that are poorly documented. Many other configuration parameters related to security and HTTP handlers are addressed in Chapters 9 and 20, respectively.

Finally, with regard to changes for ASP.NET 2.0, a handful of new attributes have been added to the `<sessionState>` settings in *web.config*, making it easier, for example, to control the behavior of cookies. Additionally, ASP.NET 2.0 offers new classes for reading, accessing, and modifying the contents of *web.config*. Beyond these enhancements, however, the basic strategies for working with *machine.config* and *web.config* are little changed from ASP.NET 1.x.

PREV

Recipe 12.2. Overriding Default HTTP Runtime Parameters in web.config

Problem

You want to change the default HTTP runtime settings for your application, such as the execution timeout setting.

Solution

Modify the *web.config* file by adding ASP.NET HTTP runtime settings to it:

1. Locate the *web.config* file in the root directory of your application (or create one if it does not already exist).
2. Add an `<httpRuntime>` element and set the `executionTimeout` and other attributes required for your application:

```
<?xml version="1.0"?>
  <configuration>
    <system.web>
   <httpRuntime executionTimeout="90"
     maxRequestLength="4096"
     useFullyQualifiedRedirectUrl="false" />
 </system.web>
</configuration>
```

Discussion

It can be useful to modify the default HTTP runtime settings in *web.config* so, for example, users of your application can upload large files. Another, perhaps more important, motivation for this recipe is to demonstrate unobtrusively how you can override the predefined settings for your application by adding elements, such as the `<httpRuntime>` element, to the default *web.config* file Visual Studio creates.

The following is a description of the attributes we've used with the `<httpRuntime>` element, which are the most commonly used attributes:

`executionTimeout`

> The `executionTimeout` attribute of `<httpRuntime>` defines the maximum amount of time in seconds that a request is allowed to run before it is automatically terminated by ASP.NET. The default value is 90 seconds. If your application has requests that take longer, such as a long-running database query, you can increase the value. The value can be any positive integer value (1 to 2,147,483,647), but large numbers are not practical.

`maxRequestLength`

> The `maxRequestLength` attribute defines the maximum size of a file that can be uploaded by the application. The value is in KB (kilobytes) and has a default value of `4096` (4MB). If your application needs to support uploading files larger than 4MB, you can change the value as required. The valid range is 0 to 2,147,483,647.
>
> Denial-of-service attacks can be launched by initiating the upload of many large files simultaneously. Therefore, the `maxRequestLength` should be set as small as possible to meet the needs of your application.

`useFullyQualifiedRedirectUrl`

> The `useFullyQualifiedRedirectUrl` attribute is a flag indicating if fully qualified URLs should be used when ASP.NET performs a redirection. Setting the value to `false` (the default) configures ASP.NET to use relative URLs (e.g., */ASPNetCookbook/ProblemMenu.aspx*) for client redirects. Setting the value to true configures ASP.NET to use fully qualified URLs (e.g., *http://localhost/ASPNetCookbook/ProblemMenu.aspx*) for all client redirects. If you are working with mobile applications, some devices will require fully qualified URLs.

The `<httpRuntime>` element contains other attributes, including those that provide control over threads used by your application and the number of requests allowed to be queued before requests are rejected. Consult the Microsoft documentation on the `<httpRuntime>` element for full details of these attributes.

See Also

MSDN documentation on the `httpRuntime` element (search for "httpRuntime element")

Recipe 12.3. Adding Custom Application Settings in web.config

Problem

You have custom configuration information for your application that you would like to store in its *web.config* file.

Solution

Modify the *web.config* file for your application by adding an `<appSettings>` element to contain the custom configuration settings:

1. Locate the *web.config* file in the root directory of your application (or create one if it does not already exist).
2. Add an `<appSettings>` element.
3. Add `<add>` child elements along with `key/value` pairs to the `<appSettings>` element as required.
4. In the code-behind class for your ASP.NET page, use the .NET language of your choice to acces the `<appSettings>` key/value collection through the `ConfigurationManager` object.

Examples 12-1, 12-2, 12-3 through 12-4 show the sample code we've written to implement this solution. Example 12-1 shows our *web.config* file with some `key/value` pairs. Example 12-2 shows the *.aspx* file for a web form that displays the configuration settings. Examples 12-3 (VB) and 12-4 (C#) show the code-behind class that accesses the configuration settings using the `ConfigurationManager` object.

Discussion

ASP.NET lets you add and then access configuration information specific to your application to the *web.config* file by means of a special `<appSettings>` element. You can add application configuration information by adding an `<add>` child element for each parameter, setting the `key` attribute to the name of the configuration parameter, and setting the `value` attribute to the value of the configuration parameter, as shown in Example 12-1.

The `<appSettings>` element is not a child of `<system.web>`, like some of the other *web.config* elements we discuss in this chapter. Rather, it is a subsection all its own within the `<configuration>` section.

When your application is started, ASP.NET creates a `NameValueCollection` from the `key/value` pairs in the `<appSettings>` section. You can access this `NameValueCollection` anywhere in your application through the `ConfigurationManager` object. Any data that can be represented as a string can be stored in the `<appSettings>` section, but anything other than a string will need to be cast to the appropriate data type for use in your application.

web.config allows any string for the value of a `key/value` pair. If the data is any nonstring data type, your code needs to include the appropriate exception handling for invalid data in the *web.config* file.

See Also

Recipe 12.6 for how to store an application's configuration information in its own custom section in *web.config*

Example 12-1. Application settings in web.config

```
<?xml version="1.0"?>
<configuration>

  <appSettings>
  <!-- Keys/Values used in chapter 12 -->
  <add key="defaultSortField" value="Title" />
  <add key="defaultSortOrder" value="Ascending" />
  <add key="defaultResultsPerPage" value="25" />
  </appSettings>

</configuration>
```

Example 12-2. Accessing application settings in web.config (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH12GetAppSettingsVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH12GetAppSettingsVB"
  Title="Get Application Settings" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">
     Application Configuration In web.config (VB)
  </div>
   <table width="60%" align="center" border="0">
    <tr>
    <td class="labelText">Sort Field: </td>
    <td class="labelText">
       <asp:Label ID="labSortField" Runat="server" /></td>
 </tr>
 <tr>
       <td class="labelText">Sort Order: </td>
    <td class="labelText">
          <asp:Label ID="labSortOrder" Runat="server" /></td>
 </tr>
 <tr>
       <td class="labelText">Number of Pages: </td>
    <td class="labelText">
       <asp:Label ID="labNumberOfPages" Runat="server" /></td>
 </tr>
  </table>
</asp:Content>
```

Example 12-3. Accessing application settings in web.config code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Configuration

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH12GetAppSettingsVB.aspx
 ''' </summary>
 Partial Class CH12GetAppSettingsVB
 Inherits System.Web.UI.Page
 '''*********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>

 Private Sub Page_Load(ByVal sender As Object, _
       ByVal e As System.EventArgs) Handles Me.Load
   Dim resultsPerPage As Integer

    'initialize labels on form from values in web.config
    labSortField.Text = ConfigurationManager.AppSettings("defaultSortField")
    labSortOrder.Text = ConfigurationManager.AppSettings("defaultSortOrder")

    'get an integer value from web.config and do a little calculating
    resultsPerPage = _
      CInt(ConfigurationManager.AppSettings("defaultResultsPerPage"))
    labNumberOfPages.Text = Math.Ceiling(1234.0 / resultsPerPage).ToString()
  End Sub 'Page_Load
 End Class 'CH12GetAppSettingsVB
End Namespace
```

Example 12-4. Accessing application settings in web.config code-behind (.cs)

```
using System;
using System.Configuration;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
  /// CH12GetAppSettingsCS.aspx
  /// </summary>
  public partial class CH12GetAppSettingsCS : System.Web.UI.Page
  {
    ///***********************************************************************
    /// <summary>
    /// This routine provides the event handler for the page load event.
    /// It is responsible for initializing the controls on the page.
    /// </summary>
    ///
    /// <param name="sender">Set to the sender of the event</param>
    /// <param name="e">Set to the event arguments</param>
    protected void Page_Load(object sender, EventArgs e)
    {
  int resultsPerPage;

  // initialize labels on form from values in web.config
  labSortField.Text =
   ConfigurationManager.AppSettings["defaultSortField"];
  labSortOrder.Text =
   ConfigurationManager.AppSettings["defaultSortOrder"];

        // get an integer value from web.config and do a little calculating
      resultsPerPage =
   Convert.ToInt32(ConfigurationManager.AppSettings["defaultResultsPerPage"]);
      labNumberOfPages.Text = Math.Ceiling(1234.0 / resultsPerPage).ToString();
  } // Page_Load
    } // CH12GetAppSettingsCS
}
```

Recipe 12.4. Displaying Custom Error Messages

Problem

You want to replace the generic messages ASP.NET displays whenever anapplication error occurs with your own custom error messages.

Solution

Create a *web.config* file, add the custom errors element to it, and create the custom error pages.

1. Locate the *web.config* file in the root directory of your application (or create one if it does not exist).
2. Add a `<customErrors>` element to the *web.config* file and add an `<error>` child element for each custom error page you want to display.
3. Create the custom error pages.

Example 12-5 shows some settings that we've added to a web.config file to demonstrate this solution.

Discussion

By default, ASP.NET displays its own error page when any of the standard server errors occurs, such as 401 (access denied), 404 (page not found), or 500 (internal server error). But a default ASP.NET error page will not match the look and feel of your application and may not provide the information you want to convey to your users. ASP.NET provides the ability, via the*web.config* file, to output your own custom error pages. A similar capability is available in IIS, but customizing the*web.config* file is simpler. Because the customization is done in the*web.config* file, moving it to another server is as simple as copying the *web.config* file and the customerror pages to the new location.

First, add a `<customErrors>` element to your *web.config* file as a child of `<system.web>`. The `mode` attribute defines when and where the custom error pages are displayed. Set the`mode` to `RemoteOnly` to have the customerror pages displayed only when accessing the application from a remote machine. When set to `RemoteOnly`, the ASP.NET error pages will not be displayed when accessing the application from the local machine. Set the`mode` to `On` to have the custom error messages displayed on local and remote machines. Set the `mode` to `Off` to display the ASP.NET error messages on local and remote machines.

The `defaultRedirect` attribute defines the custom error page that will be displayed if an error occurs and there is no specific error element (described later) for the error. By default the`defaultRedirect` attribute is set to `GenericErrorPage.htm`. You should change this to your own generic error page.

Next, add an `error` element for each server error you want to redirect to a custom error page. Set the `statusCode` attribute to the server error code, and set the `redirect` attribute to the URL of the page to be displayed when the error occurs. You can include parameters in the URL if desired.

When the error is a 404 error (page not found), for example, ASP.NET includes a parameter in the URL to indicate the name of the requested page that was not found. The URL for the redirection of th 404 error described would be:

```
http://[server]/ASPNetCookbook/PageNotAvailable.aspx?aspxerrorpath=
/ASPNetCookbook/BadPage.aspx
```

Your application can use the `Request.QueryString` collection to retrieve the name of the page that was not found and include the information in your custom page:

```
labMessage.Text = Request.QueryString("aspxerrorpath") & _
     " Is Not Available On This Site"

labMessage.Text = Request.QueryString["aspxerrorpath"] +
     " Is Not Available On This Site";
```

See Also

Search "`<customErrors>` element" in the MSDN library.

Example 12-5. Custom error settings in web.config

```
<?xml  version="1.0"?>
<configuration>
   <system.web>

     <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">
     <error statusCode="404" redirect="PageNotAvailable.aspx"/>
  </customErrors>

   </system.web>
</configuration>
```

PREV

Recipe 12.5. Maintaining Session State Across Multiple Web Servers

Problem

You need to configure your application to maintain session state acrossmultiple web serversfor example, in a load-balanced web farm or web garden.

Solution

When the data stored in session is easy to re-create or is not critical, configure your application to use the ASP.NET State Service using the following four steps:

1. Set up a new server with the .NET Framework installed to maintainsession state on behalf of your application.
2. Start the ASP.NET State Service on the designated machine.
3. Modify the application *web.config* file on its current web server, as shown next, setting the `<sessionState>` element's `mode` attribute to `StateServer` and `StateConnectionString` attribute to the IP address and port of the state server:

   ```
   <sessionState
    mode="StateServer"
     stateConnectionString="tcpip=10.0.1.11:42424"
    cookieless="false"
    timeout="20"  />
   ```

4. Copy the contents of the root folder and subfolders of your application on the current web server to the additional web servers.

When you must not lose any session information if server problems arise, use SQL Server to store all session information, as follows:

1. In the instance of SQL Server that you will use for this purpose, install the special tables and stored procedures that ASP.NET requires by running the *InstallSqlState.sql* script provided with the .NET Framework located in the *%SystemRoot%\Microsoft.NET\Framework\%VersionNumber%* folder.

2. Set up a database user with read/write access to the `tempDB` database.

3. Modify the *web.config* file, as shown next, setting the IP address to the address of your SQL Server machine and replacing *`user`* and *`pwd`* with the settings for the database user with read/write access to the `tempDB` database:

```
<sessionState
 mode="SQLServer"
 sqlConnectionString="data source=10.0.1.12;user id=  user ;password=pwd "
 cookieless="false"
 timeout="20" />
```

Discussion

ASP.NET provides in-process session management much like classic ASP. However, scaling to multiple servers in classic ASP has always been difficult because session information is not shared between ASP servers. There are ways to solve the problem in classic ASP but all are hard and code changes are generally required when you move to a multiple-server configuration.

ASP.NET provides two methods for managing session state when you decide to deploy a web application on multiple servers: the ASP.NET State Service and support for session state storage in SQL Server.

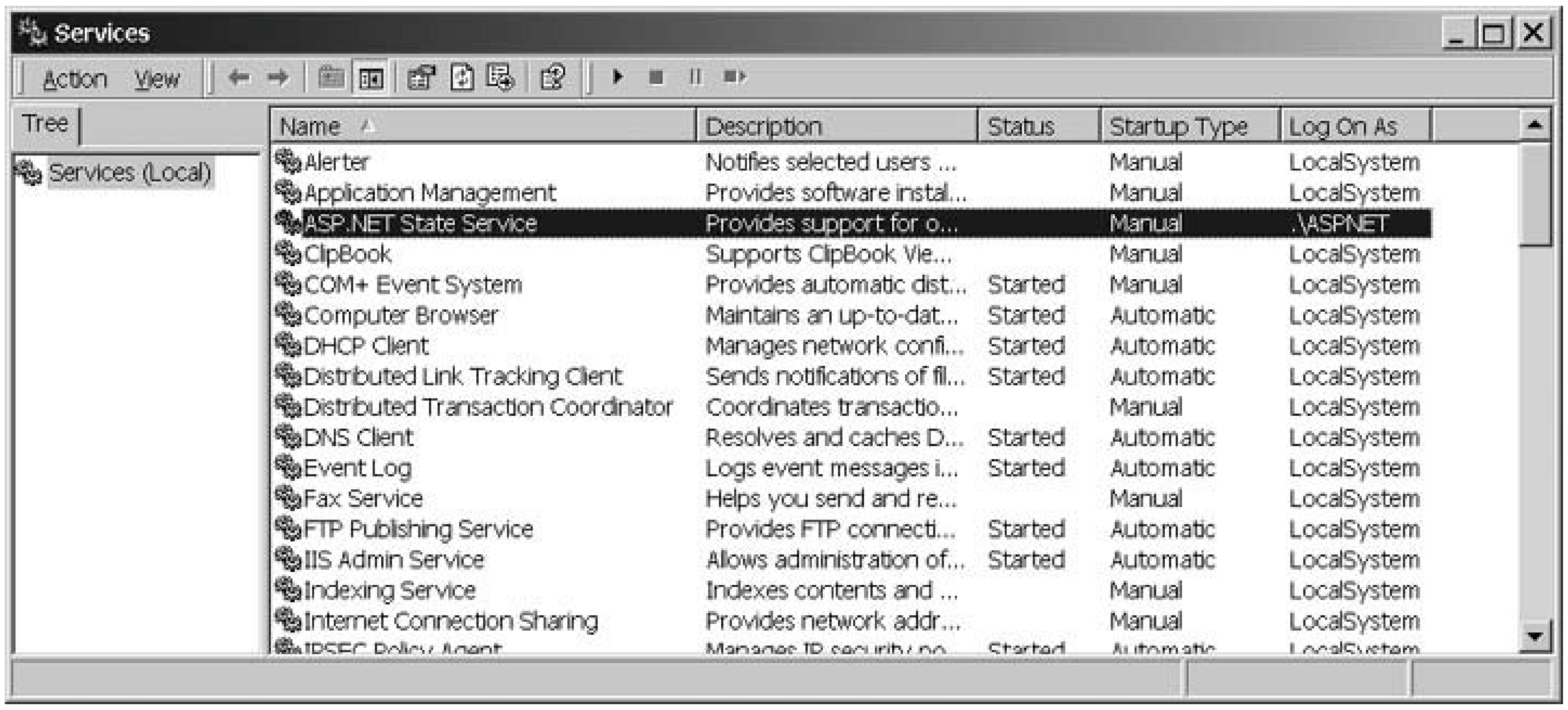
The ASP.NET State Service is an out-of-process, memory-based session management service that is intended to be used when the data stored in session is easy to recreate or is not critical. The SQL Server-based session management uses SQL Server to store all session information and is intended to be used when you must not lose any session information if server problems arise. As you might expect, the performance of SQL Server in maintaining session state management is worse than the ASP.NET State Service because of the overhead involved in querying and writing session information in a database. The ASP.NET State Service will have a poorer performance as the default in-process storage of an ASP.NET application because of the out-of-process communications and network hops involved in communicating with other servers across a network.

The out-of-process State Service and SQL Server session management techniques are compatible with each other and require no changes to your code when you move from one technique to another, which you might do, for example, if you are losing critical session information.

If you choose to make use of the ASP.NET State Service, install and run it on a server separate from the web servers in the web farm. This allows any one of the web servers to be taken down for maintenance or replacement without interrupting access to your application. If the ASP.NET State Service is running on a server running an instance of the application, when that server requires maintenance, your entire application will be unavailable while the server is down.

You start the ASP.NET State Service the same way you start any other Windows service. Invoke the Services console by clicking on Start   Control Panel and selecting Administrative Tools and then Services. In the Services console window, select the ASP.NET State Service, as shown in Figure 12-1
.

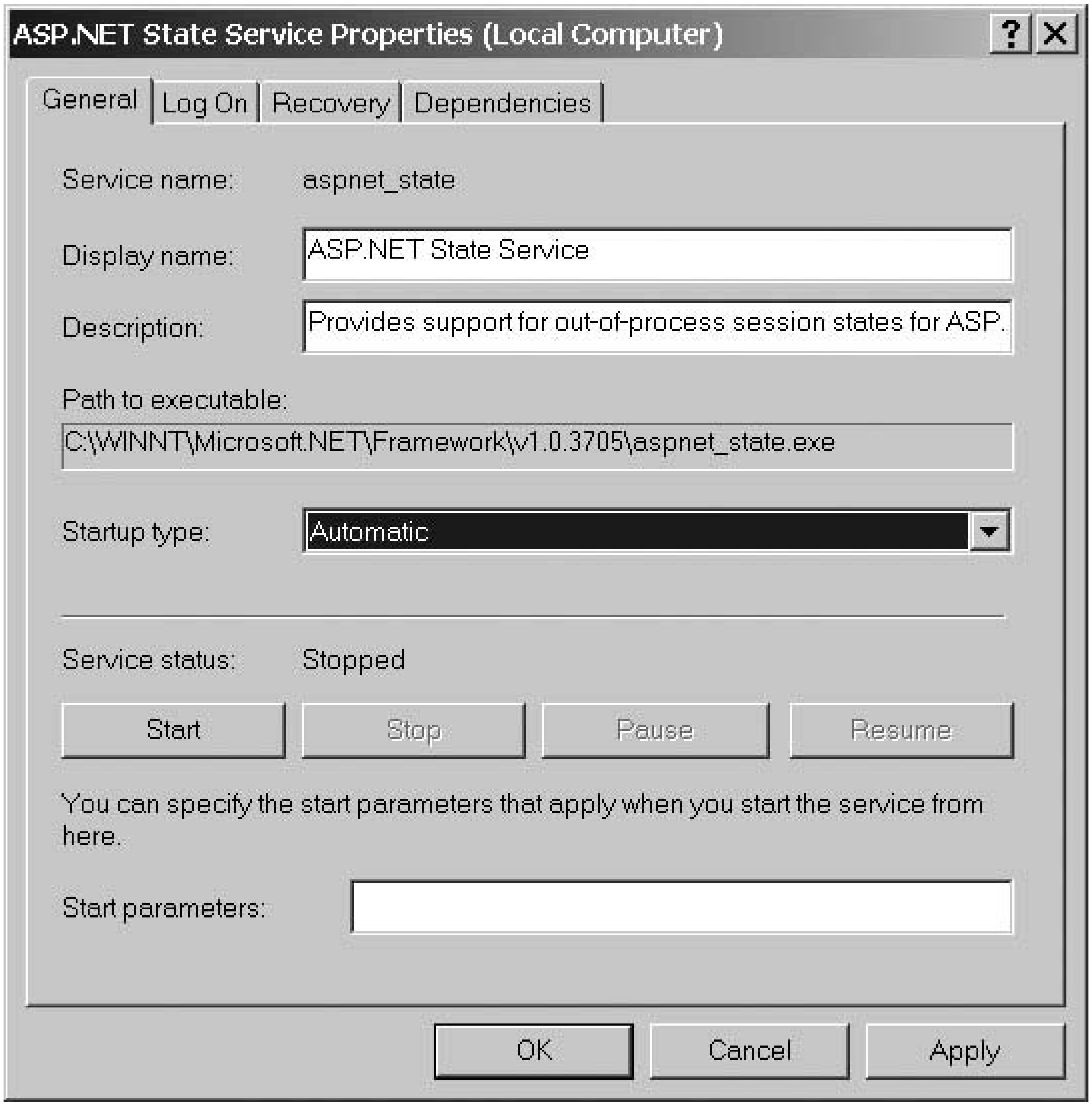
Figure 12-1. Starting the ASP.NET State Service



Name	Description	Status	Startup Type	Log On As
Alerter	Notifies selected users ...		Manual	LocalSystem
Application Management	Provides software instal...		Manual	LocalSystem
ASP.NET State Service	Provides support for o...		Manual	.\ASPNET
ClipBook	Supports ClipBook Vie...		Manual	LocalSystem
COM+ Event System	Provides automatic dist...	Started	Manual	LocalSystem
Computer Browser	Maintains an up-to-dat...	Started	Automatic	LocalSystem
DHCP Client	Manages network confi...	Started	Automatic	LocalSystem
Distributed Link Tracking Client	Sends notifications of fil...	Started	Automatic	LocalSystem
Distributed Transaction Coordinator	Coordinates transactio...		Manual	LocalSystem
DNS Client	Resolves and caches D...	Started	Automatic	LocalSystem
Event Log	Logs event messages i...	Started	Automatic	LocalSystem
Fax Service	Helps you send and re...		Manual	LocalSystem
FTP Publishing Service	Provides FTP connecti...	Started	Automatic	LocalSystem
IIS Admin Service	Allows administration of...	Started	Automatic	LocalSystem
Indexing Service	Indexes contents and ...		Manual	LocalSystem
Internet Connection Sharing	Provides network addr...		Manual	LocalSystem
IPSEC Policy Agent	Manages IP security po...	Started	Automatic	LocalSystem

Right-click the ASP.NET State Service and select Properties to open the ASP.NET State Service Properties dialog box. From the Startup type drop-down menu, select Automatic to start the service automatically any time the server is restarted, and then click the Start button to start the service, as shown in Figure 12-2 .

Figure 12-2. Setting the ASP.NET State Service's Startup type property tc Automatic


ASP.NET State Service Properties (Local Computer)

General | Log On | Recovery | Dependencies

Service name: aspnet_state

Display name: ASP.NET State Service

Description: Provides support for out-of-process session states for ASP.

Path to executable:
C:\WINNT\Microsoft.NET\Framework\v1.0.3705\aspnet_state.exe

Startup type: Automatic

Service status: Stopped

Start | Stop | Pause | Resume

You can specify the start parameters that apply when you start the service from here.

Start parameters:

OK | Cancel | Apply

To use the ASP.NET State Service, you must make two changes to the application*web.config* file on its current web server. First, add a `<sessionState>` element to the *web.config* file in the root directory of your application (unless the element is present). Next, set (or change) the`mode` attribute to `"StateServer"` and modify the `stateConnectionString` attribute to include the IP address of the server running the ASP.NET State Service. We'll refer to that machine as the*state server* from here on. Do not change the port number because this is the port the State Service is listening on. If you must change the port number, you will need to make changes to the registry settings for the State Service because the port number cannot be changed through the Services console. Here's some sample code that demonstrates the changes we've described:

```
<?xml version="1.0"?>
<configuration>
 <system.web>

   …
```

```
  <sessionState mode="StateServer"
      stateConnectionString="tcpip=10.0.1.11:42424"
      cookieless="false"
      timeout="20" />

   …
  </system.web>
 </configuration>
```

In addition to modifying `<sessionState>` settings in *web.config* , you will need to alter the keys used for encryption and generation of unique values. Session management uses these keys to generate session IDs and hash values used in the session collection, so all servers must be configured to use the same values. If the values differ from server to server, the session information will not be understood by all of the servers hosting the application. You control encryption through the `<machineKey>` element, which you will need to add to the *web.config* file. By default, the keys are set to generate values automatically that will result in a different key being used on each server. These values should be set to 40128 hexadecimal characters as a function of the encryption type and level. Note that [*YourHexValue*] must be replaced with the hexadecimal value you want to use for your application:

```
 <?xml  version="1.0"?>
 <configuration   xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>

   …

  <machineKey validationKey="[YourHexValue]"
       decryptionKey="[YourHexValue]"
       validation="SHA1"/>

       …

  </system.web>
 </configuration>
```

After you've tested your application with the ASP.NET state server enabled, you're ready to deploy it to the other servers.

An alternate approach to using the ASP.NET state server is to use SQL Server to store and persist the session information. ASP.NET again makes this a simple configuration change with no code changes required.

The first step to using SQL Server for session management is to install the special tables and stored procedures that ASP.NET requires in the instance of SQL Server that you will use for this purpose. You create this support by running the *InstallSqlState.sql* script provided with the .NET Framework.

To run the script, open a command prompt and navigate to the folder *\WINNT\Microsoft.NET\Framework\<version>* , where *<version>* is the version of the .NET Framework installed on your SQL Server machine. At the command prompt, type the following, and all of the necessary tables, stored procedures, and the like will be installed. You will need to substitute for `password` the appropriate password for the `sa` account:

```
OSQL -S localhost -U sa -P password < InstallSqlState.sql
```

If you need to uninstall the SQL Server state management objects, you will have to use the same command line. Substitute `UninstallSqlState.sql` for `InstallSqlState.sql` .

For a production application, set up a database user with read/write access to the `tempDB` database. You will use this database user in the configuration described next.

The only other change required to use SQL Server for state management is to modify the *web.config* file, as shown here, setting the `mode` to `SQLServer` , setting the IP address to the address of your SQL Server machine, and replacing `user` and `pwd` with the settings for the database user with read/write access to the `tempDB` database:

```
<sessionState mode="SQLServer"
  sqlConnectionString="data source=10.0.1.12;user id=user;password=pwd"
  cookieless="false"
  timeout="20" />
```

From now on, when your application runs, it will use SQL Server for storing all of its session information. You can see the results of this by looking at the `ASPStateTempApplications` and `ASPStateTempSessions` tables in the `tempDB` database. You should find entries there for your applications and sessions, respectively. Because the data is stored in binary format, you will not be able to view it.

Two other `sessionState` element attributes are frequently used in conjunction with the other modifications that we've discussed: `cookieless` and `timeout` .

`cookieless`

Defines whether cookies are used to store the session ID and the behavior of the cookies. Possible values are `UseCookies, UseUri, AutoDetect` , and `UseDeviceProfile` . `UseCookies` specifies that cookies will always be used to store the session ID. `UseUri` specifies the session ID will always be placed in the URL (e.g., *http://localhost/ASPNetCookbook/(3eqccw45tlzqa555hakqrn55)/Home.aspx*). `AutoDetect` specifies cookies will be used if the requesting device supports cookies and cookies are enablec

If cookies are not supported or not enabled, the URL will be used for the session ID. UseDeviceProfile specifies the storage location for the session ID is determined by the capabilities of the requesting device. This is similar to AutoDetect except no attempt is made to determine if cookies are enabled. The determination is made by the device support. The defaul value is UseDeviceProfile for backward compatibility with ASP.NET 1.x. (This attribute is new for ASP.NET 2.0.)

`timeout`

The `timeout` attribute is used to define the length of inactivity in a session before the session is terminated. The value is in minutes and is `20` by default.

> You may be aware that you can configure the session timeout in IIS. In an ASP.NET application, however, the session timeout that can be configured in IIS is not used.

See Also

The article by Rob Howard titled "ASP.NET Session State (Nothin' but ASP.NET)" in the MSDN library (search "ASP.NET Session State")

PREV

Recipe 12.6. Accessing Other web.config Configuration Elements

Problem

You want to be able to read application information from a *web.config* file that is unavailable as an `<appSettings>` `key/value` pair but present as an attribute or child element of some other element of the file.

Solution

Use the `OpenWebConfiguration` method of the `WebConfigurationManager` object to read the *web.config* file into a `Configuration` object, and use the `GetSection` method to access the desired section.

In the code-behind class for your ASP.NET page, use the .NET language of your choice to:

1. Use the `OpenWebConfiguration` method of the `WebConfigurationManager` object to read the *web.config* file into a `Configuration` object.
2. Use the `GetSection` method to access the desired section.
3. Cast the returned `ConfigurationSection` object to the type of the object for the section being accessed.
4. Use the properties of the object to access the desired information.

Examples 12-6, 12-7 through 12-8 show an application we've written that implements this solution and retrieves attribute settings from the `<trace>` element of a *web.config* file. Example 12-6 shows the *.aspx* file that displays the information. Examples 12-7 (VB) and 12-8 (C#) show the code-behind class for the page that does the work of reading the settings from the `<trace>` element.

Discussion

In ASP.NET 1.x, you had to resort to opening the *web.config* file using an `XmlDocument` object to access many of the configuration elements. In ASP.NET 2.0, new classes have been introduced to read and access *web.config*. The new classes provide the ability to change the contents of the configuration file and save it back to the filesystem. In addition, the new classes are strongly typed, which avoids the problem of storing data of the wrong type in the *web.config* file.

The first step is to open the *web.config* file by using the `OpenWebConfiguration` method of the

WebConfigurationManager object. The relative path within your web site to the *web.config* file is passed to OpenWebConfiguration and a Configuration object containing the contents of the *web.config* file is returned.

VB

```
Dim config As Configuration
Dim path As String

path = Request.ApplicationPath
config = WebConfigurationManager.OpenWebConfiguration(path)
```

C#

```
Configuration config = null;
String path = null;

path = Request.ApplicationPath;
config = WebConfigurationManager.OpenWebConfiguration(path);
```

Any *web.config* file contained in your application can be accessed by passing the file's relative path to OpenWebConfiguration.

Next, use the GetSection method of the Configuration object to access the desired section. Since the GetSection method returns a ConfigurationSection object, you will need to cast it to the type of the section you are accessing. In our example, we are accessing the trace section, so we cast the ConfigurationSection object to a traceSection object. This allows us to access the data in the trace section using strongly typed properties.

```
Dim traceSettings As TraceSection

traceSettings = CType(config.GetSection("system.web/trace"), _
     TraceSection)

TraceSection traceSettings = null;

traceSettings = (TraceSection)(config.GetSection("system.web/trace"));
```

Finally, use the properties of the object to access the desired information. In our example, we are displaying the values of a few of the attributes in the trace section.

```
labEnabled.Text = traceSettings.Enabled.ToString()
```

```
labRequestLimit.Text = traceSettings.RequestLimit.ToString()
labPageOutput.Text = traceSettings.PageOutput.ToString()
labTraceMode.Text = traceSettings.TraceMode.ToString()
labLocalOnly.Text = traceSettings.LocalOnly.ToString()
```

```
labEnabled.Text = traceSettings.Enabled.ToString();
labRequestLimit.Text = traceSettings.RequestLimit.ToString();
labPageOutput.Text = traceSettings.PageOutput.ToString();
labTraceMode.Text = traceSettings.TraceMode.ToString();
labLocalOnly.Text = traceSettings.LocalOnly.ToString();
```

If you need to modify the contents of the *web.config* file, you can change the values of the desired properties and use the `Save` method of the `Configuration` object:

```
traceSettings.Enabled = Not traceSettings.Enabled
traceSettings.PageOutput = Not traceSettings.PageOutput
config.Save()
```

```
traceSettings.Enabled = !traceSettings.Enabled;
traceSettings.PageOutput = !traceSettings.PageOutput;
config.Save();
```

You should not design your application to change the contents of the *web.config* file routinely. The reason is that changing the *web.config* file's contents may cause the application to restart. Restarting the application can lead to a significant delay for page requests and can cause undesirable effects for users when their sessions are restarted.

Example 12-6. Access system configuration information in web.config (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH12AccessSystemConfigVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH12AccessSystemConfigVB"
  Title="Access web.config" %>

<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
    Accessing System Configuration Information In web.config (VB)
 </div>
 <table width="40%" align="center" border="0">
    <tr>
        <td align="center" class="subHeading" colspan="2>
       Trace Section Attributes
    </td>
    </tr>
    <tr>
     <td align="right" width="50%" class="labelText">enabled = </td>
     <td width="50%" class="labelText">
    <asp:Label ID="labEnabled" runat="server" />
    </td>
    </tr>
    <tr>
    <td align="right" class="labelText">requestLimit = </td>
    <td class="labelText">
       <asp:Label ID="labRequestLimit" runat="server" />
    </td>
    </tr>
    <tr>
    <td align="right" class="labelText">pageOutput = </td>
    <td class="labelText">
    <asp:Label ID="labPageOutput" runat="server" />
    </td>
    </tr>
    <tr>
    <td align="right" class="labelText">traceMode = </td>
    <td class="labelText">
    <asp:Label ID="labTraceMode" runat="server" />
    </td>
    </tr>
    <tr>
    <td align="right" class="labelText">localOnly = </td>
    <td class="labelText">
    <asp:Label ID="labLocalOnly" runat="server" />
    </td>
    </tr>
    <tr>
     <td align="center" colspan="2">
    <br/>
    <input id="btnToggleEnable" runat="server"
       type="button" value="Toggle Enable"
       onserverclick="btnToggleEnable_ServerClick" />
    </td>
    </tr>
  </table>
</asp:Content>
```

Example 12-7. Access system configuration information in web.config code-behind (.vb)

```
Option Explicit
On Option Strict On

Imports System
Imports System.Configuration
Imports System.Web.Configuration

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH12AccessSystemConfigVB.aspx
 ''' </summary>
 Partial  Class  CH12AccessSystemConfigVB
   Inherits System.Web.UI.Page
   '''*********************************************************************
   ''' <summary>
   ''' This routine provides the event handler for the page load event. It
   ''' is responsible for initializing the controls on the page.
   ''' </summary>
   '''
   ''' <param name="sender">Set to the sender of the event</param>
   ''' <param name="e">Set to the event arguments</param>
   Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
  Dim config As Configuration
  Dim traceSettings As TraceSection
  Dim path As String

  If (Not Page.IsPostBack) Then
     path = Request.ApplicationPath
     config = WebConfigurationManager.OpenWebConfiguration(path)

     traceSettings = CType(config.GetSection("system.web/trace"), _
         TraceSection)

     initializeForm(traceSettings)
  End If
  End Sub 'Page_Load

  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the toggle enable button
  ''' click event. It is responsible for toggling the trace enable state
  ''' and updating the data in the web.config file.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnToggleEnable_ServerClick(ByVal sender As Object, _
            ByVal e As System.EventArgs)
  Dim config As Configuration
  Dim traceSettings As TraceSection
  Dim path As String

  path = Request.ApplicationPath
  config = WebConfigurationManager.OpenWebConfiguration(path)

  traceSettings = CType(config.GetSection("system.web/trace"), _
      TraceSection)
  traceSettings.Enabled = Not traceSettings.Enabled
  traceSettings.PageOutput = Not traceSettings.PageOutput
  config.Save()

  initializeForm(traceSettings)
  End Sub 'btnToggleEnable_ServerClick

  '''*************************************************************************
  ''' <summary>
  ''' This routine is responsible for displaying the initializing the
  ''' controls on the form with the passed trace setting information
  ''' </summary>
  '''
  ''' <param name="traceSettings">Set to the TraceSection information to
  ''' be used to initialize the controls on the form
  ''' </param>
  Private Sub initializeForm(ByVal traceSettings As TraceSection)
  labEnabled.Text = traceSettings.Enabled.ToString()
  labRequestLimit.Text = traceSettings.RequestLimit.ToString()
  labPageOutput.Text = traceSettings.PageOutput.ToString()
  labTraceMode.Text = traceSettings.TraceMode.ToString()
  labLocalOnly.Text = traceSettings.LocalOnly.ToString()
  End Sub 'initializeForm
  End Class 'CH12AccessSystemConfigVB
End Namespace
```

Example 12-8. Access system configuration information in web.config code-behind (.cs)

```
using System;
using System.Configuration;
using System.Web.Configuration;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH12AccessSystemConfigCS.aspx
 /// </summary>
 public partial class CH12AccessSystemConfigCS : System.Web.UI.Page
 {
   ///***********************************************************************
   /// <summary>
   /// This routine provides the event handler for the page load event.
   /// It is responsible for initializing the controls on the page.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void Page_Load(object sender, EventArgs e)
   {
   Configuration config = null;
   TraceSection traceSettings = null;
   String path = null;

   if (!Page.IsPostBack)
   {
   path = Request.ApplicationPath;
   config = WebConfigurationManager.OpenWebConfiguration(path);

   traceSettings = (TraceSection)(config.GetSection("system.web/trace"));
   initializeForm(traceSettings);
  }
  } // Page_Load

  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the toggle enable button
  /// click event. It is responsible for toggling the trace enable state
  /// and updating the data in the web.config file.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnToggleEnable_ServerClick(Object sender,
            EventArgs e)
  {
  Configuration config = null;
  TraceSection traceSettings = null;
  String path = null;

  path = Request.ApplicationPath;
  config = WebConfigurationManager.OpenWebConfiguration(path);

  traceSettings = (TraceSection)(config.GetSection("system.web/trace"));
  traceSettings.Enabled = !traceSettings.Enabled;
  traceSettings.PageOutput = !traceSettings.PageOutput;
  config.Save();
```

```
  initializeForm(traceSettings);
  } // btnToggleEnable_ServerClick

  ///***********************************************************************
  /// <summary>
  /// This routine is responsible for displaying the initializing the
  /// controls on the form with the passed trace setting information
  /// </summary>
  ///
  /// <param name="traceSettings">Set to the TraceSection information to
  /// be used to initialize the controls on the form
  /// </param>
  private void initializeForm(TraceSection traceSettings)
  {
  labEnabled.Text = traceSettings.Enabled.ToString();
  labRequestLimit.Text = traceSettings.RequestLimit.ToString();
  labPageOutput.Text = traceSettings.PageOutput.ToString();
  labTraceMode.Text = traceSettings.TraceMode.ToString();
  labLocalOnly.Text = traceSettings.LocalOnly.ToString();

  } // initializeForm
  } // CH12AccessSystemConfigCS
}
```

PREV

Recipe 12.7. Adding Your Own Configuration Elements to web.config

Problem

You want to create and add your own configuration elements to *web.config* . No predefined element will do, nor will use of the `<appSettings> key/value` entries of *web.config* , as described in Recipe 12.2.

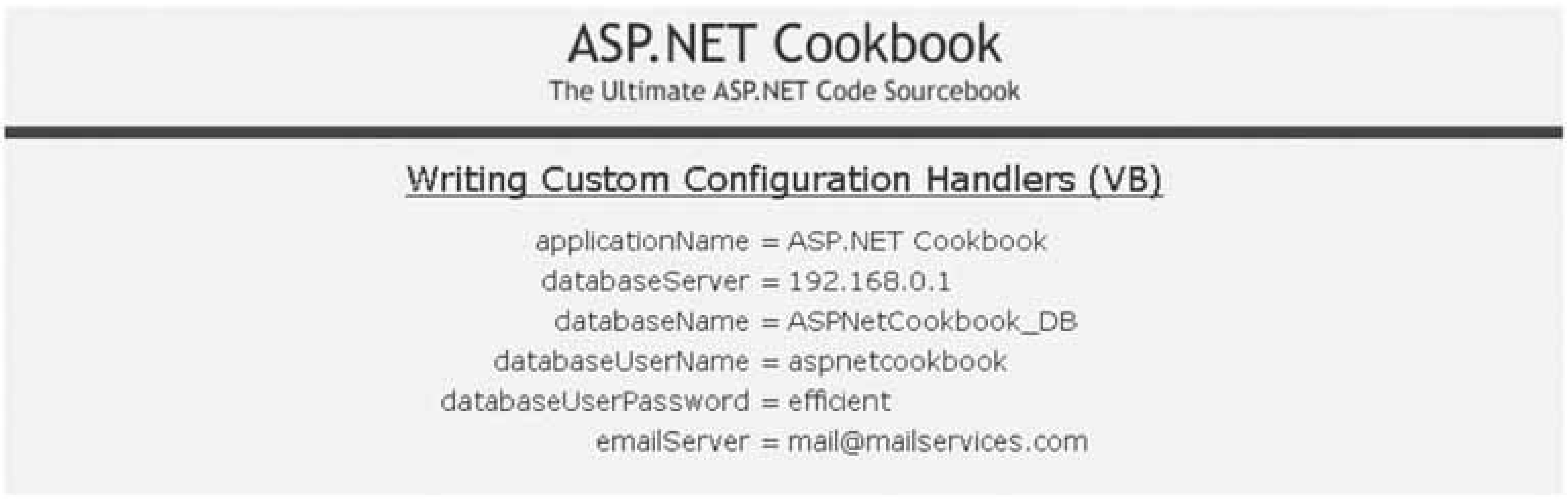
Solution

Determine what configuration information you want to store in *web.config* , create your own custom section handler for parsing the element, add the definition of the section handler to *web.config* , add the new configuration element to *web.config* , and use the configuration information in your application.

1. Determine what configuration information you want to store in *web.config* .
2. Use the .NET language of your choice to create a custom section handler class for parsing your newly defined element.
3. Add the definition of the section handler to the `<configSections>` section of *web.config* .
4. Add the configuration element to *web.config* and assign values to its attributes or child elements.
5. In the code-behind class for your ASP.NET page, use the .NET language of your choice to acces and put the custom configuration data to work.

The code we've written to illustrate this solution appears in Examples 12-9, 12-10 , 12-11 , 12-12 , 12-13 through 12-14 . Examples 12-9 (VB) and 12-10 (C#) show the class for a custom section handler. The changes we've made to *web.config* to have it use the custom section handler are shown in Example 12-11 . Example 12-12 shows the *.aspx* file for a sample web form that displays some custom configuration settings. Examples 12-13 and 12-14 show the code-behind class that accesses the custom configuration information. The output of the sample web form showing the data read usir the custom configuration handler is shown in Figure 12-3.

Figure 12-3. Configuration information read by configuration handler


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Writing Custom Configuration Handlers (VB)

```
      applicationName = ASP.NET Cookbook
       databaseServer = 192.168.0.1
         databaseName = ASPNetCookbook_DB
     databaseUserName = aspnetcookbook
 databaseUserPassword = efficient
          emailServer = mail@mailservices.com
```

Discussion

Sometimes the predefined configuration elements provided by ASP.NET, including the `key/value` collection available through the `<appSettings>` element of *web.config* (described in Recipe 12.2), are not enough. When this is the case, being able to store any configuration information required by you application in its own custom section in *web.config* can be a useful alternative. What's more, having a custom section in *web.config* can be a real boon to program clarity and ease of implementation.

As a prelude to discussing the custom section for *web.config* , it's worthwhile reviewing the basic structure of a *machine.config* file, as shown next, because it is the *machine.config* file whose structure we will mimic. The root node of the XML document is `<configuration>` . The `<configSections>` child element is next. It defines the configuration section elements that will follow and the configuration handlers (*classes*) that will handle the configuration information contained in the elements. When `<configSections>` is present, it must be the first child element of `<configuration>` .

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

 <configSections>

   …

 </configSections>

 <!-- sections defined in configSections -->

</configuration>
```

Even if you are used to looking at a *web.config* file, you have probably never seen the `<configSections>` element. This is because all of the standard configuration handlers are defined in *machine.config* . (ASP.NET reads the *machine.config* file and then the *web.config* file(s) for your application. All of the data from the *machine.config* file is used unless a section in *web.config* overrides the equivalent section in *machine.config* .) Because `<configSections>` is only used to define configuration handlers, it never appears in a *web.config* file unless your application uses custom configuration handlers like the one described in this example.

In this example, we use a `<siteProperties>` custom element to hold the information we want to store in *web.config* and that will be used by our custom configuration handler. We have chosen the path of using a custom element (and a custom configuration handler) because of the flexibility it affords us and the clarity with which we can express the information to be stored.

In creating your own custom element, you first need to determine the configuration information you want to store in *web.config* and then define the format you want it stored in. The only limitation is that the data element must be a well-formed XML node. You can use attributes, child elements, or any combination of the two to hold your custom configuration information. In our example, we use attributes only. The well-formed XML that defines the section we want to add to our application *web.config* is shown here:

```
<siteProperties applicationName="ASP.NET Cookbook"
       databaseServer="192.168.0.1"
      databaseName="ASPNetCookbook_DB"
      databaseUserName="aspnetcookbook"
      databaseUserPassword="efficient"
        emailServer="mail@mailservices.com" />
```

The section can be named anything you like as long as it does not conflict with any predefined ASP.NET elements. We recommend that, for consistency, you stick with the mixed-case convention used by ASP. NET. Section names and attributes are case-sensitive.

After defining your configuration element and specifying the values of the information it will store, create a custom configuration handler in a separate project. This way, the created assembly can be reused easily in multiple applications. For our example, the project was named *VBCustomConfigHandlers* (*CSCustomConfigHandlers* for C#), which by default will generate an assembly with the same name.

In your configuration handler project, add a new class named to reflect the configuration element the handler supports. In our example, we've named the class `SiteConfigHandlerVB` (`SiteConfigHandlerCS` for C#) because the section added to *web.config* contains site configuration information.

To act as a custom configuration handler, the class must implement the `IConfigurationSectionHandler` interface:

```
Namespace VBCustomConfigHandlers

Public Class SiteConfigHandlerVB
  Implements IConfigurationSectionHandler

    …
  End Class SiteConfigHandlerVB
End Namespace 'VBCustomConfigHandlers
```

C#

```
namespace CSCustomConfigHandlers
  {
  public class SiteConfigHandlerCS : IConfigurationSectionHandler
 {

  …

 } // SiteConfigHandlerCS
 } // CSCustomConfigHandlers
```

The IConfigurationSectionHandler interface requires that you implement a single method, Create , which requires three parameters: parent, configContext , and section . The parent parameter provides a reference to the corresponding parent configuration section, and the configContext parameter provides a reference to the current ASP.NET context. Neither is used in this example. The section parameter provides a reference to the XML node (section) of the *web.config* file that is to be processed. In our example, section will be set to reference the <siteProperties> section added to *web.config* .

```
 Namespace VBCustomConfigHandlers
  Public Class VBSiteConfigHandler
    Implements IConfigurationSectionHandler

    Public Function Create(ByVal parent As Object, _
         ByVal configContext As Object, _
      ByVal section As XmlNode) As Object _

     Implements IConfigurationSectionHandler.Create

   …

    End Function 'Create
  End Class 'VBSiteConfigHandler
 End Namespace 'VBCustomConfigHandlers


 namespace CSCustomConfigHandlers
 {
  public class CSSiteConfigHandler : IConfigurationSectionHandler
  {

   public Object Create(Object parent,
         Object configContext,
      XmlNode section)
   {
    …
   } // Create
  } // CSSiteConfigHandler
 } // CSCustomConfigHandlers
```

The `Create` method returns an object that contains the configuration information from the passed section. This can be anything you want it to be. The most flexible approach is to define a class that will contain the data and provide easy access by your application. In our example, the returned class is defined in the same file as the `SiteConfigHandlerVB` (`SiteConfigHandlerCS` for C#) class.

VB

```
 Namespace VBCustomConfigHandlers
  Public Class VBSiteConfigHandler
   Implements IConfigurationSectionHandler

   Public Function Create(ByVal parent As Object, _
        ByVal configContext As Object, _
        ByVal section As XmlNode) As Object _

       Implements IConfigurationSectionHandler.Create

      …

      End Function 'Create
     End Class 'VBSiteConfigHandler

  'The following class provides the container returned by
  'the SiteConfigHandlerVB
  Public Class SiteConfigurationVB

   …

  End Class 'SiteConfigurationVB
 End Namespace 'VBCustomConfigHandlers


 namespace CSCustomConfigHandlers
 {
  public class CSSiteConfigHandler : IConfigurationSectionHandler
  {
   public Object Create(Object parent,
     Object configContext,
     XmlNode section)
   {
    …

   } // Create
  } // CSSiteConfigHandler

  // The following class provides the container returned by
  // the SiteConfigHandlerCS
  public class SiteConfigurationCS
  {
   …
  } // SiteConfigurationCS

 } // CSCustomConfigHandlers
```

The `SiteConfigurationVB (SiteConfigurationCS` for C#) class needs a constructor that has parameters for each of the configuration items and a read-only property for each of theconfiguration items. The code for our example is shown in Examples 12-9(VB) and 12-10 (C#).

After the class that will be used for the return is defined, the`Create` method should extract the configuration information from the passed XML section, create a new instance of the `SiteConfigurationVB (SiteConfigurationCS` for C#) object, and return a reference to the new instance. For our example, attributes were used for the configuration information in the section. This allows us to get a reference to the attributes collection of the section and to extract the individual values by using the `GetNamedItem` method of the attributes collection.

```
 Public Function Create(ByVal parent As Object, _
       ByVal configContext As Object, _
       ByVal section As XmlNode) As Object _

     Implements IConfigurationSectionHandler.Create

 Dim siteConfig As SiteConfigurationVB
 Dim attributes As XmlAttributeCollection

 attributes = section.Attributes
 With attributes
  siteConfig = _
  New SiteConfigurationVB(.GetNamedItem("applicationName").Value, _
     .GetNamedItem("databaseServer").Value, _
     .GetNamedItem("databaseName").Value, _
     .GetNamedItem("databaseUserName").Value, _
     .GetNamedItem("databaseUserPassword").Value, _
     .GetNamedItem("emailServer").Value)
 End With 'attributes

 Return (siteConfig)
  End Function 'Create


 public Object Create(Object parent,
     Object configContext,
     XmlNode section)
 {
  SiteConfigurationCS siteConfig = null;
  XmlAttributeCollection attributes = null;

  attributes = section.Attributes;
  siteConfig =
   new SiteConfigurationCS(
    attributes.GetNamedItem("applicationName").Value,
    attributes.GetNamedItem("databaseServer").Value,
    attributes.GetNamedItem("databaseName").Value,
    attributes.GetNamedItem("databaseUserName").Value,
    attributes.GetNamedItem("databaseUserPassword").Value,
    attributes.GetNamedItem("emailServer").Value);
  return (siteConfig);
 } // Create
```

With the customconfiguration handler, you need to add the handler information to*web.config* to tell ASP.NET how to handle the `<siteProperties>` section you've added. Add a `<configSections>` element at the top of your *web.config* that contains a single `section` element. The `name` attribute defines the name of the custom section containing your configuration information. The`type` attribute defines the class and assembly name in the form `type` = *`"class, assembly"`* that will process your custom configuration section. The *`class`* name must be a fully qualified class name. The*`assembly`* name must be the name of the assembly (`dll`) created in your configuration handler project, described earlier.

```
<configSections>
<section name="siteProperties"
  type="ASPNetCookbook.VBCustomConfigHandlers.SiteConfigHandlerVB,
     VbCustomConfigHandlers" />
</configSections>
```

When the `<configSections>` element is present, it must be the first element in the *web.config* file after the `<configuration>` element or a parsing exception will be thrown.

The last thing you need to do before you can use the custom configuration in your application is to add a reference to the `VBCustomConfigHandlers` (`CSCustomConfigHandlers` for C#) assembly in your application.

Accessing the custom configuration information in your application requires using `ConfigurationManager.GetConfig` and passing it the name of your custom section. This method returns an object that must be cast to the object type you returned in your custom configuration handler class. After the reference is obtained, all of the site information is available as properties of the object.

VB

```
Dim siteConfig As SiteConfigurationVB

siteConfig = CType(ConfigurationManager.GetSection("siteProperties"), _
      SiteConfigurationVB)
labApplicationName.Text = siteConfig.applicationName
labDBServer.Text = siteConfig.databaseServer
labDBName.Text = siteConfig.databaseName
labDBUserName.Text = siteConfig.databaseUserName
labDBUserPassword.Text = siteConfig.databaseUserPassword
labEmailServer.Text = siteConfig.emailServer
```

C#

```
SiteConfigurationCS siteConfig = null;

siteConfig = (SiteConfigurationCS)
    (ConfigurationManager.GetSection("siteProperties"));
labApplicationName.Text = siteConfig.applicationName;
labDBServer.Text = siteConfig.databaseServer;
labDBName.Text = siteConfig.databaseName;
labDBUserName.Text = siteConfig.databaseUserName;
labDBUserPassword.Text = siteConfig.databaseUserPassword;
labEmailServer.Text = siteConfig.emailServer;
```

See Also

Recipe 12.2

Example 12-9. Custom section handler class (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Configuration
Imports System.Xml

Namespace ASPNetCookbook.VBCustomConfigHandlers
  ''' <summary>
  ''' This class provides a custom site configuration handler.
  ''' </summary>
  Public Class SiteConfigHandlerVB
  Implements IConfigurationSectionHandler

   '''*************************************************************************
     ''' <summary>
  ''' This routine provides the creation of the SiteConfigurationVB object
  ''' from the passed section of the web.config file
  ''' </summary>
  '''
  ''' <param name="parent">Set to the parent of this section</param>
  ''' <param name="configContext">Set the HttpContext of the current
  ''' request
  ''' </param>
  ''' <param name="section">Set to the XML node containing the
  ''' configuration data to be used to create this object
  ''' </param>
  ''' <returns>SiteConfigHandlerVB initialied from the passed section
  ''' of the web.config file
  ''' </returns>
  Public Function Create(ByVal parent As Object, _
                ByVal configContext As Object, _
         ByVal section As XmlNode) As Object _

    Implements IConfigurationSectionHandler.Create

  Dim siteConfig As SiteConfigurationVB
  Dim attributes As XmlAttributeCollection

  attributes = section.Attributes
  With attributes
     siteConfig = _
   New SiteConfigurationVB(.GetNamedItem("applicationName").Value, _
      .GetNamedItem("databaseServer").Value, _
      .GetNamedItem("databaseName").Value, _
      .GetNamedItem("databaseUserName").Value, _
      .GetNamedItem("databaseUserPassword").Value, _
      .GetNamedItem("emailServer").Value)

  End With 'attributes
     Return (siteConfig)
   End Function 'Create
 End Class 'SiteConfigHandlerVB

 'The following class provides the container returned by
 'the SiteConfigHandlerVB
Public Class SiteConfigurationVB
 Private mApplicationName As String
 Private mDatabaseServer As String
 Private mDatabaseName As String
 Private mDatabaseUserName As String
 Private mDatabaseUserPassword As String
 Private mEmailServer As String

 Public ReadOnly Property applicationName() As String
  Get
   Return (mApplicationName)
  End Get
 End Property 'applicationName

    Public ReadOnly Property databaseServer() As String
  Get
   Return (mDatabaseServer)
  End Get
 End Property 'databaseServer

    Public ReadOnly Property databaseName() As String
  Get
   Return (mDatabaseName)
  End Get
 End Property 'databaseName

 Public ReadOnly Property databaseUserName() As String
  Get
   Return (mDatabaseUserName)
  End Get
 End Property 'databaseUserName

    Public ReadOnly Property databaseUserPassword() As String
  Get
   Return (mDatabaseUserPassword)
  End Get
 End Property 'databaseUserPassword

    Public ReadOnly Property emailServer() As String
  Get
   Return (mEmailServer)
  End Get
 End Property 'emailServer

   '''*********************************************************************
   ''' <summary>
   ''' This constructor creates the object and populates the attributes
   ''' with the passed values
   ''' </summary>
   Public Sub New(ByVal applicationName As String, _
      ByVal databaseServer As String, _
      ByVal databaseName As String, _
      ByVal databaseUserName As String, _
      ByVal databaseUserPassword As String, _
      ByVal emailServer As String)

  mApplicationName = applicationName
  mDatabaseServer = databaseServer
  mDatabaseName = databaseName
  mDatabaseUserName = databaseUserName
  mDatabaseUserPassword = databaseUserPassword
  mEmailServer = emailServer
   End Sub 'New
 End Class 'SiteConfigurationVB
End Namespace
```

Example 12-10. Custom section handler class (.cs)

```
using System;
using System.Configuration;
using System.Xml;

namespace ASPNetCookbook.CSCustomConfigHandlers
{
  /// <summary>
  /// This class provides a custom site configuration handler.
  /// </summary>
  public class SiteConfigHandlerCS : IConfigurationSectionHandler
  {
   ///*************************************************************************
  /// <summary>
  /// This routine provides the creation of the SiteConfigurationVB object
  /// from the passed section of the web.config file
  /// </summary>
  ///
  /// <param name="parent">Set to the parent of this section</param>
  /// <param name="configContext">Set the HttpContext of the current
  /// request
  /// </param>
  /// <param name="section">Set to the XML node containing the
  /// configuration data to be used to create this object
  /// </param>
  /// <returns>SiteConfigHandlerVB initialied from the passed section
  /// of the web.config file
  /// </returns>

  public Object Create(Object parent,
    Object configContext,
    XmlNode section)
  {
   SiteConfigurationCS siteConfig = null;
   XmlAttributeCollection attributes = null;

   attributes = section.Attributes;
   siteConfig =
    new SiteConfigurationCS(
    attributes.GetNamedItem("applicationName").Value,
    attributes.GetNamedItem("databaseServer").Value,
    attributes.GetNamedItem("databaseName").Value,
    attributes.GetNamedItem("databaseUserName").Value,
    attributes.GetNamedItem("databaseUserPassword").Value,
    attributes.GetNamedItem("emailServer").Value);

   return (siteConfig);
  } // Create
 } // SiteConfigHandlerCS

 // The following class provides the container returned by
 // the SiteConfigHandlerCS
 public class SiteConfigurationCS
 {
  private String mApplicationName = null;
  private String mDatabaseServer = null;
  private String mDatabaseName = null;
  private String mDatabaseUserName = null;
  private String mDatabaseUserPassword = null;
  private String mEmailServer = null;
 public String applicationName
 {
  get
  {
   return (mApplicationName);
  }
 } // applicationName
 public String databaseServer
 {
  get
  {
   return (mDatabaseServer);
  }
 } // databaseServer

 public String databaseName
 {
  get
  {
   return (mDatabaseName);
  }
 } // databaseName

 public String databaseUserName
 {
  get
  {
   return (mDatabaseUserName);
  }
 } // databaseUserName

 public String databaseUserPassword
 {
  get
  {
   return (mDatabaseUserPassword);
  }
 } // databaseUserPassword

 public String emailServer
 {
  get
  {
   return (mEmailServer);
  }
 } // emailServer

 //*************************************************************************
 //
 //   ROUTINE: Constructor
 //
 //   DESCRIPTION: This constructor creates the object and populates the
 //      attributes with the passed values
 //-------------------------------------------------------------------------
 public SiteConfigurationCS(String applicationName,
       String databaseServer,
       String databaseName,
       String databaseUserName,
       String databaseUserPassword,
       String emailServer)
 {
  mApplicationName = applicationName;
  mDatabaseServer = databaseServer;
  mDatabaseName = databaseName;
  mDatabaseUserName = databaseUserName;
  mDatabaseUserPassword = databaseUserPassword;
  mEmailServer = emailServer;
 } // SiteConfigurationCS
  } // SiteConfigurationCS
}
```

Example 12-11. Changes to web.config to use the custom section handler

```
<?xml version="1.0"?>
<configuration>
<configSections>
  <section name="siteProperties"
  type="ASPNetCookbook.VBCustomConfigHandlers.SiteConfigHandlerVB,
     VBCustomConfigHandlers" />

</configSections>

<system.web>

 …

</system.web>

<!-- Setting for custom configuration section handler in chapter 12
  -->
<siteProperties applicationName="ASP.NET Cookbook"
    databaseServer="192.168.0.1"
    databaseName="ASPNetCookbook_DB"
    databaseUserName="aspnetcookbook"
    databaseUserPassword="efficient"
    emailServer="mail@mailservices.com" />
</configuration>
```

Example 12-12. Sample web form using custom configuration data (.aspx

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH12CustomConfigHandlerVB.aspx.vb"
   Inherits="ASPNetCookbook.VBExamples.CH12CustomConfigHandlerVB"
  Title="Custom Config Handler" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">
     Writing Custom Configuration Handlers (VB)
  </div>
  <table width="60%" align="center" border="0">
     <tr>
     <td align="right" width="50%" class="labelText">
   applicationName = </td>
     <td width="50%" class="labelText">
        <asp:Label ID="labApplicationName" runat="server" />
    </td>
  </tr>
  <tr>
    <td align="right" class="labelText">
   databaseServer = </td>
    <td class="labelText">
   <asp:Label ID="labDBServer" runat="server" />
    </td>
  </tr>
  <tr>
    <td align="right" class="labelText">
      databaseName = </td>
    <td class="labelText">
   <asp:Label ID="labDBName" runat="server" />
    </td>
  </tr>
  <tr>
     <td align="right" class="labelText">
     databaseUserName = </td>
     <td class="labelText">
     <asp:Label ID="labDBUserName" runat="server" />
     </td>
  </tr>
  <tr>
     <td align="right" class="labelText">
        databaseUserPassword = </td>
     <td class="labelText">
     <asp:Label ID="labDBUserPassword" runat="server" />
    </td>
  </tr>
     <tr>
     <td align="right" class="labelText">
        emailServer = </td>
     <td class="labelText">
        <asp:Label ID="labEmailServer" runat="server" />
     </td>
  </tr>
 </table>
</asp:Content>
```

Example 12-13. Sample web form using custom configuration data code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports ASPNetCookbook.VBCustomConfigHandlers
Imports System
Imports System.Configuration

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 '''  CH12CustomConfigHandlerVB.aspx
 ''' </summary>
 Partial Class CH12CustomConfigHandlerVB
  Inherits System.Web.UI.Page
   '''**************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
          ByVal e As System.EventArgs) Handles Me.Load

    Dim siteConfig As SiteConfigurationVB

    siteConfig = CType(ConfigurationManager.GetSection("siteProperties"), _
     SiteConfigurationVB)
    labApplicationName.Text = siteConfig.applicationName
    labDBServer.Text = siteConfig.databaseServer
    labDBName.Text = siteConfig.databaseName
    labDBUserName.Text = siteConfig.databaseUserName
    labDBUserPassword.Text = siteConfig.databaseUserPassword
    labEmailServer.Text = siteConfig.emailServer

  End Sub 'Page_Load
 End Class 'CH12CustomConfigHandlerVB
End Namespace
```

Example 12-14. Sample web form using custom configuration data code-behind (.cs)

```
using ASPNetCookbook.CSCustomConfigHandlers;
using System;
using System.Configuration;
using System.Web.Configuration;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code behind for
   /// CH12CustomConfigHandlerCS.aspx
  /// </summary>
   public partial class CH12CustomConfigHandlerCS : System.Web.UI.Page
  {
   ///**************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   SiteConfigurationCS siteConfig = null;

   siteConfig = (SiteConfigurationCS)
     (ConfigurationManager.GetSection("siteProperties"));
   labApplicationName.Text = siteConfig.applicationName;
   labDBServer.Text = siteConfig.databaseServer;
   labDBName.Text = siteConfig.databaseName;
   labDBUserName.Text = siteConfig.databaseUserName;
   labDBUserPassword.Text = siteConfig.databaseUserPassword;
   labEmailServer.Text = siteConfig.emailServer;
  } // Page_Load
 } // CH12CustomConfigHandlerCS
}
```

PREV

Recipe 12.8. Encrypting web.config Sections

Problem

You have sensitive data in your *web.config* file, such as the connection string used to access your database, that you do not want available in plain text.

Solution

Use the Protected Configuration feature to encrypt the sensitive information stored in*web.config*:

1. Add the sensitive information to your *web.config*, such as a `<connectionStrings>` element:

```
<configuration>
 <connectionStrings>
  <add name="sqlConnectionString"
   connectionString="Data Source=localhost;
   Initial Catalog=ASPNetCookbook;
    UID=ASPNetCookbook_User;PWD=w0rk;
    persist security info=False;Connection Timeout=30;" />
 </connectionStrings>

 …

</configuration>
```

2. Add a `<machineKey>` element to your *web.config*:

```
<configuration>

 …

  <system.web>
   <machineKey validationKey="AutoGenerate,IsolateApps"
    decryptionKey="AutoGenerate,IsolateApps" />
  </system.web>
</configuration>
```

3. Run the *aspnet_regiis.exe* tool to encrypt the sensitive data element, such as the `<connectionStrings>` element with the following command:

```
aspnet_regiis -pe "connectionStrings" -app "[Your Application Name]"
```

4. Run the *aspnet_regiis.exe* tool to encrypt the `<machineKey>` element:

```
aspnet_regiis -pe "system.web/machineKey" -app "[Your Application Name]"
```

5. Run the *aspnet_regiis.exe* tool to grant access to the key container by the ASP.NET identity:

```
aspnet_regiis -pa "NetFrameworkConfigurationKey" "[ASP.NET User]"
```

Discussion

Applications frequently contain sensitive data in their*web.config* files, such as a database connection string that contains the credentials required to access the database. If this information is stored in your *web.config* in plain text, anyone who gains access to the web.config file will have the credentials to access the database for your application.

ASP.NET 1.x had no provisions for storing sensitive data in*web.config* other than in plain text. ASP.NET 2.0 provides the ability to encrypt sensitive data in*web.config* and to decrypt the data automatically when needed by your application without requiring any changes to your code.

The first step in encrypting elements in *web.config* is to add the element, such as `<connectionStrings>`, and test your application to ensure the data has been entered correctly. This is important since once the data is encrypted it cannot be changed without it first being decrypted (decrypting is discussed later).

Next, you need to add a `<machineKey>` element to *web.config*. The `<machineKey>` element configures the keys used for forms authentication cookie data, view state data, and managing the encrypted element(s). The attributes of the `<machineKey>` element are described as follows:

`decryptionKey`

The `decryptionKey` attribute defines the key that will be used for encryption and decryption or the process that will be used to generate the keys. The value can be set to`"AutoGenerate"`, `"AutoGenerate,IsolateApps"`, or a string of hexadecimal characters.

"AutoGenerate,IsolateApps" is the default. When set to "AutoGenerate", ASP.NET generates a randomkey. When the value is set to "AutoGenerate,IsolateApps", ASP.NET generates a unique key for each application using each application's ID. Setting the value to a hexadecimal string allows you to control the key and is required if your application runs on multiple servers where the keys must be identical on each server.

decryption

The decryption attribute defines the hashing algorithmused for encrypting/decrypting the data. The value can be Auto, AES, or 3DES. The default is Auto.

validationKey

The validationKey attribute defines the key that will be used for validation of encrypted data. The validationKey is not used for encryption/decryption of *web.config* elements. It is used to generate the message authentication code (MAC) when the enableViewStateMAC attribute is set to true. The value can be set to "AutoGenerate", "AutoGenerate,IsolateApps", or a string of hexadecimal characters. "AutoGenerate,IsolateApps" is the default. When set to "AutoGenerate", ASP.NET generates a random key. When the value is set to "AutoGenerate,IsolateApps", ASP.NET generates a unique key for each application using each application's ID. Setting the value to a hexadecimal string will allow you to control the key and will be required if your application runs on multiple servers where the keys must be identical on each server.

validation

The validation attribute defines the type of encryption used for validating data. The value can be AES, MD5, SHA1, or tripleDES. The default is SHA1.

Once *web.config* is set up, the *aspnet_regiis.exe* tool is used to performthe data encryption. Open a command prompt and change to the *%SystemRoot%\Microsoft.NET\Framework\%VersionNumber%* folder where the *aspnet_regiis.exe* tool is located. To encrypt the <connectionStrings> element in *web.config*, execute the following command in the command window, substituting the virtual path of your application for [*Your Application Name*].

```
aspnet_regiis -pe "connectionStrings" -app "[Your Application Name]"
```

The <connectionStrings> element of *web.config* will be changed with the sensitive data encrypted, as follows. (The CipherValue elements have been shortened for clarity.)

```
<configuration   xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  k<connectionStrings
   configProtectionProvider="RsaProtectedConfigurationProvider">
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
      xmlns="http://www.w3.org/2001/04/xmlenc#">
         <EncryptionMethod
```

```
     Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
     <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
       <EncryptionMethod
       Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
       <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <KeyName>Rsa Key</KeyName>
       </KeyInfo>
       <CipherData>
      <CipherValue>VJyz/bFoxgJU2PWl…..</CipherValue>
        </CipherData>
     </EncryptedKey>
   </KeyInfo>
   <CipherData>
    <CipherValue>Bb/JK94rxfCphYQebP3s…..</CipherValue>
   </CipherData>
  </EncryptedData>
   </connectionStrings>

 …

</configuration>
```

The next step is to encrypt the `<machineKey>` element to protect the key information. To encrypt the `<machineKey>` element, execute the following command in the command window, substituting the virtual path of your application for [*Your Application Name*].

```
 aspnet_regiis -pe "system.web/machineKey" -app "[Your Application Name]"
```

The `<machineKey>` element of *web.config* file will be changed with the key data encrypted as follows. (The `CipherValue` elements have been shortened for clarity.)

```
 <configuration   xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">

  …

   <system.web>
      <machineKey configProtectionProvider="RsaProtectedConfigurationProvider">
      <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
     xmlns="http://www.w3.org/2001/04/xmlenc#">
     <EncryptionMethod
     Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
     <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
     <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
    <EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <KeyName>Rsa Key</KeyName>
    </KeyInfo>
    <CipherData>
    <CipherValue>aVVNMATLnm48…..</CipherValue>
    </CipherData>
    </EncryptedKey>
   </KeyInfo>
   <CipherData>
    <CipherValue>f9/2H0sgeSeZIC/t…..</CipherValue>
   </CipherData>
    </EncryptedData>
    </machineKey>

  …

  </system.web>
   </configuration>
```

The final step to encrypting data in *web.config* is to grant access to the key container for the ASP.NET user. The *aspnet_regiis.exe* tool is once again used with the following command, substituting the ASP.NET user account on your server for [*ASP.NET User*]:

```
aspnet_regiis -pa "NetFrameworkConfigurationKey" "[ASP.NET User]"
```

By default, the ASP.NET user is `ASPNET` when using Windows 2000 Server and `NT AUTHORITY\NETWORK SERVICE` when using Windows 2003 Server. If you have configured ASP.NET to use another user, you will need to use that user account. The name of the ASP.NET user can be determined by creating an ASP.NET page with the following content and displaying the page in a browser:

```
<%@ Page Language="VB" %>
<%
Response.Write(System.Security.Principal.WindowsIdentity.
GetCurrent().Name)
%>
```

```
<%@ Page Language="C#" %>
<%
Response.Write(System.Security.Principal.WindowsIdentity.
GetCurrent().Name);
%>
```

When determining the ASP.NET user, displaying this page while running in Visua Studio 2005 will report the user as the logged in user, not the account under which ASP.NET runs.

Granting access to the key container must be done on all servers where your application will run. If you fail to grant access, ASP.NET will not be able to decrypt the data in *web.config*.

If you need to change the encrypted data in *web.config*, you will need to decrypt the data, make the required changes, and then encrypt the data again. The decryption is performed by using the *aspnet_regiis.exe* tool, this time using `pd` command-line parameter:

```
To decrypt the <connectionStrings> element:
 aspnet_regiis -pd "connectionStrings" -app "[Your Application Name]"

To decrypt the <machineKey> element:
 aspnet_regiis -pe "system.web/machineKey" -app "[Your Application Name]"
```

See Also

"Encrypting Configuration Information Using Protected Configuration" in the MSDN Library

PREV

Chapter 13. Tracing and Debugging

PREV

PREV

13.0 Introduction

The recipes in this chapter show how you can locate problems in your ASP.NET applications by using features that support tracing and debugging.

The first seven recipes show how to use tracing to pinpoint the causes of problems in your code. *Tracing* allows you, through a configuration setting or page-level attribute, to have ASP.NET write a whole host of information about the executing request to the page or to a trace log. We start by discussing how you initiate page-and application-level tracing. We show how you can dynamically turn on page-level tracing when an exception occurs. Next, we show you how to make tracing work for componentsthose that will run on the web, as well as those that will be used elsewhere. The latte technique is important when you don't want your use of tracing-related code to preclude you from using a component outside of ASP.NET. We also show you how to write trace information to the even log and send the information via email from within a component.

Finally, we discuss *debugging*, specifically how setting conditional breakpoints can be a powerful technique for debugging your ASP.NET applications. Setting conditional breakpoints is especially useful for stopping execution at a specific point in iteration-heavy code, the focus of one of this chapter's recipes.

The recipes in this chapter are intended to help remove some of the mystery of tracing and debugging, and, in the process, help you use these techniques earlier in the development cycle when problems are often less costly to fix.

Recipe 13.2. Uncovering Page-Level Problems

Problem

You want to find the source of a problem that appears to be associated with a particular page of your application, such as a page that completes its operations more slowly than desired.

Solution

Enable page-level tracing for the page in question by setting the`trace` attribute of the `@ Page` directive in the *.aspx* file to `"TRue"` and then using `TRace.Write` (or `TRace.Warn`) statements as warranted in your code-behind to write trace information to the trace output.

Examples 13-1, 13-2 through 13-3 show the code we've written to illustrate this solution. Example 13-1 shows the *.aspx* file for a typical ASP.NET page. The code-behind class for the page appears in Examples 13-2 (VB) and 13-3 (C#). By running the page and analyzing the trace sequence, you can see how long certain key operations are taking. The output with the trace sequence is shown in Figure 13-1.

Discussion

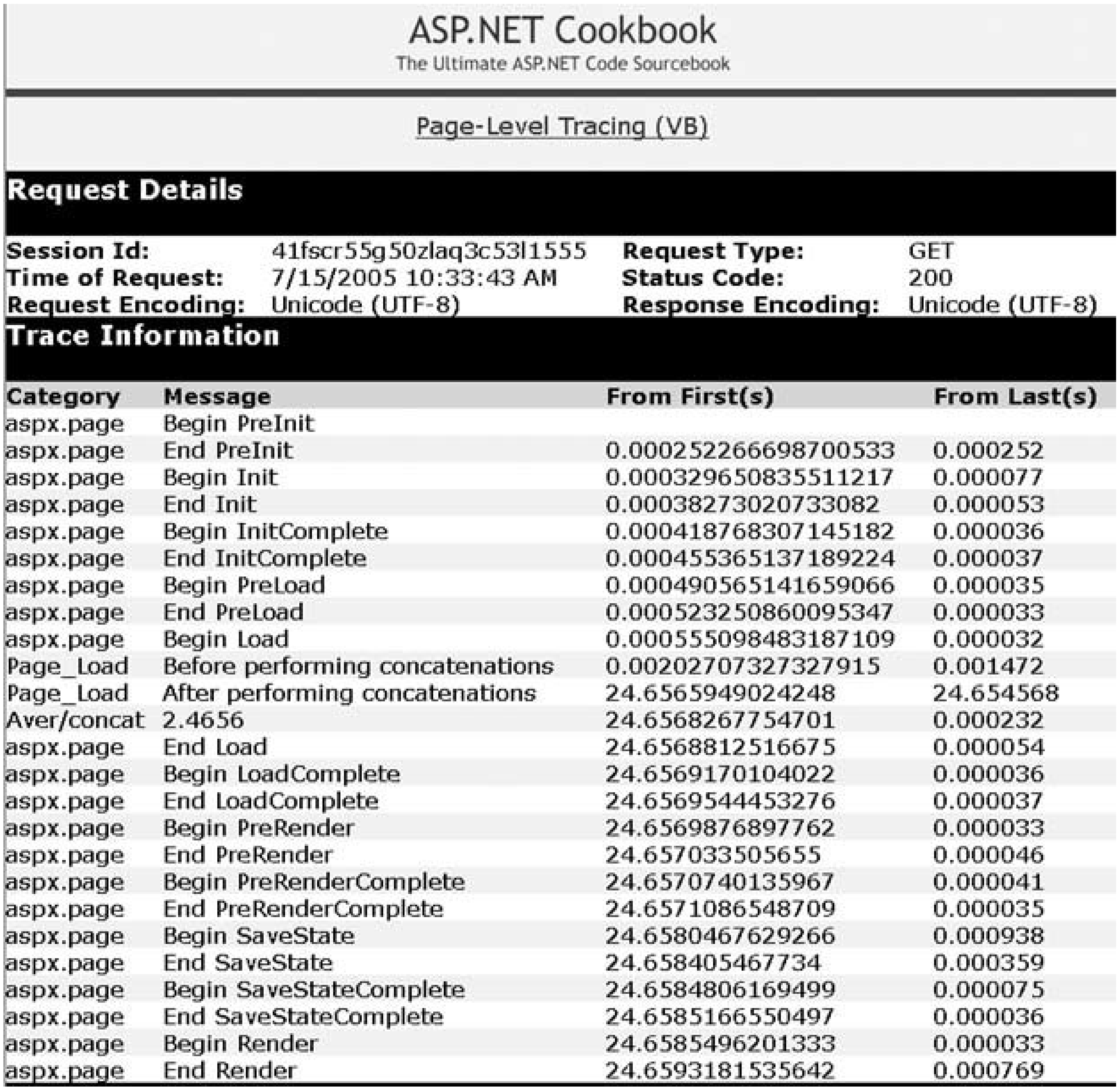
Tracing tracks and presents the execution details about an HTTP request. The`TraceContext` class is where ASP.NET stores information about an HTTP request and its trace information. You access the `TraceContext` class through the `Page.Trace` property of an ASP.NET page. To enable tracing for the page, be sure to set the `trace` attribute of the `@ Page` directive in the *.aspx* file to `"TRue"`, as shown in Example 13-1.

The `traceContext` class has two methods for writing statements into the trace log:`Write` and `Warn`. The only difference is that`Warn` outputs statements in red so they are easier to spot in the trace log. Both methods are overloaded and have three versions. If you pass a single string argument, ASP.NE will write it to the Message column of the trace log, as shown inFigure 13-1. If you use two string arguments, the first string will appear in the Category column and the second in the Message column If you use a third argument, it will have to be of type Exception and contain information about an error, which ASP.NET then writes to the trace log.

If you've placed `trace.Write` or `trace.Warn` statements in your code, you will not have to worry abou removing them later. The common language runtime (CLR) will ignore them when tracing is disabled Disable page-level tracing before deploying your application to a production environment.

In our example, `trace.Write` is used three times to put custom messages into the trace sequence: the first time to mark the start of the concatenations and the second to mark the end of the concatenations. The third message outputs the average time for a string concatenation. The latter shows how inefficient it is to use a classic concatenation operator (& or +) in ASP.NET string operations. (See Recipe 19.2 for further discussion of this code as well as the advantages of using th `StringBuilder` object to build strings over the classic concatenation operators.)

Figure 13-1. Sample tracing output



ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

Page-Level Tracing (VB)

Request Details

Session Id:	41fscr55g50zlaq3c53l1555	**Request Type:**	GET
Time of Request:	7/15/2005 10:33:43 AM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.000252266698700533	0.000252
aspx.page	Begin Init	0.000329650835511217	0.000077
aspx.page	End Init	0.00038273020733082	0.000053
aspx.page	Begin InitComplete	0.000418768307145182	0.000036
aspx.page	End InitComplete	0.000455365137189224	0.000037
aspx.page	Begin PreLoad	0.000490565141659066	0.000035
aspx.page	End PreLoad	0.000523250860095347	0.000033
aspx.page	Begin Load	0.000555098483187109	0.000032
Page_Load	Before performing concatenations	0.00202707327327915	0.001472
Page_Load	After performing concatenations	24.6565949024248	24.654568
Aver/concat	2.4656	24.6568267754701	0.000232
aspx.page	End Load	24.6568812516675	0.000054
aspx.page	Begin LoadComplete	24.6569170104022	0.000036
aspx.page	End LoadComplete	24.6569544453276	0.000037
aspx.page	Begin PreRender	24.6569876897762	0.000033
aspx.page	End PreRender	24.657033505655	0.000046
aspx.page	Begin PreRenderComplete	24.6570740135967	0.000041
aspx.page	End PreRenderComplete	24.6571086548709	0.000035
aspx.page	Begin SaveState	24.6580467629266	0.000938
aspx.page	End SaveState	24.658405467734	0.000359
aspx.page	Begin SaveStateComplete	24.6584806169499	0.000075
aspx.page	End SaveStateComplete	24.6585166550497	0.000036
aspx.page	Begin Render	24.6585496201333	0.000033
aspx.page	End Render	24.6593181535642	0.000769

In Figure 13-1 the trace log (beginning with "Request Details") appears below the standard output fo the ASP.NET page enabled for `trace`. Here's an explanation of the "Trace Information" section contents in the trace log:

Category

A custom trace category that you specified as the first argument in a`trace.Write` (or `trace.Warn`) method call.

Message

A custom trace message that you specified as the second argument in a`trace. Write` (or `TRace.Warn`) method call.

From First (s)

The time, in seconds, since the request processing was started (a running total).

From Last (s)

The time, in seconds, since the last message was displayed. This column is especially helpful for seeing how long individual operations are taking.

See Also

Recipe 19.3

Example 13-1. Page-level tracing (.aspx)

```
<%@ Page Trace="True" Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH13TestPageLevelTracingVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH13TestPageLevelTracingVB"
 Title="Test Page Level Tracing" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Page-Level Tracing (VB)
 </div>
</asp:Content>
```

Example 13-2. For page-level tracing code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH13TestPageLevelTracingVB.aspx
 ''' </summary>
 Partial  Class  CH13TestPageLevelTracingVB
  Inherits System.Web.UI.Page
  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
   Const STRING_SECTION As String = "1234567890"

   Dim testStr As String
   Dim counter As Integer
   Dim startTime As DateTime
   Dim elapsedTime As TimeSpan
   Dim loops As Integer

   'output trace message indicating the start of the concatenations
   Trace.Write("Page_Load", "Before performing concatenations")

   'Measure the elapsed time for 10,000 classic string concatenations
   loops = 10000
   startTime = DateTime.Now( )
   testStr = ""
   For counter = 1 To loops
    testStr &= STRING_SECTION
   Next

   'output trace message indicating the end of the concatenations
   Trace.Write("Page_Load", "After performing concatenations")

   'calculate the elapsed time for the string concatenations
   elapsedTime = DateTime.Now.Subtract(startTime)

   'Write average time per concatenation in milliseconds to trace sequence
   Trace.Write("Aver/concat", _
    (elapsedTime.TotalMilliseconds / loops).ToString("0.0000"))
  End Sub 'Page_Load
 End  Class  'CH13TestPageLevelTracingVB
End Namespace
```

Example 13-3. For page-level tracing code-behind (.cs)

```
using System;
using System.Configuration;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH13TestPageLevelTracingCS.aspx
 /// </summary>
 public partial class CH13TestPageLevelTracingCS : System.Web.UI.Page
 {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   const string STRING_SECTION = "1234567890";

   string testStr = null;
   DateTime startTime;
   TimeSpan elapsedTime;

   int counter;
   int loops;

   // output trace message indicating the start of the concatenations
   Trace.Write("Page_Load", "Before performing concatenations");

   // measure the elapsed time for 10000 classic string concatenations
   loops = 10000;
   startTime = DateTime.Now;
   testStr = "";
   for (counter = 1; counter <= loops; counter++)
   {
    testStr += STRING_SECTION;
   }

   // output trace message indicating the end of the concatenations
   Trace.Write("Page_Load", "After performing concatenations");
```

```
    // calculate the elapsed time for the string concatenations
    elapsedTime = DateTime.Now.Subtract(startTime);

    // Write average time per concatenation in milliseconds to trace sequence
    Trace.Write("Aver/concat",
     (elapsedTime.TotalMilliseconds / loops).ToString("0.0000"));
  } // Page_Load
 } //  CH13TestPageLevelTracingCS
}
```

PREV

PREV

Recipe 13.3. Uncovering Application-Wide Problems

Problem

You want to find the sources of problems at any point in an application, but you don't want to have t
change every page to do so, nor do you want to disrupt the output of your application pages.

Solution

Enable application-level tracing in the application *web.config* file and view the AXD application trace log for your application.

1. Locate the *web.config* file in the root directory of your application (or create one if it does not exist).
2. Enable application-level tracing by adding a `<trace>` element to the `<system.web>` section of *web.config* and setting its enabled attribute to `"true"`:

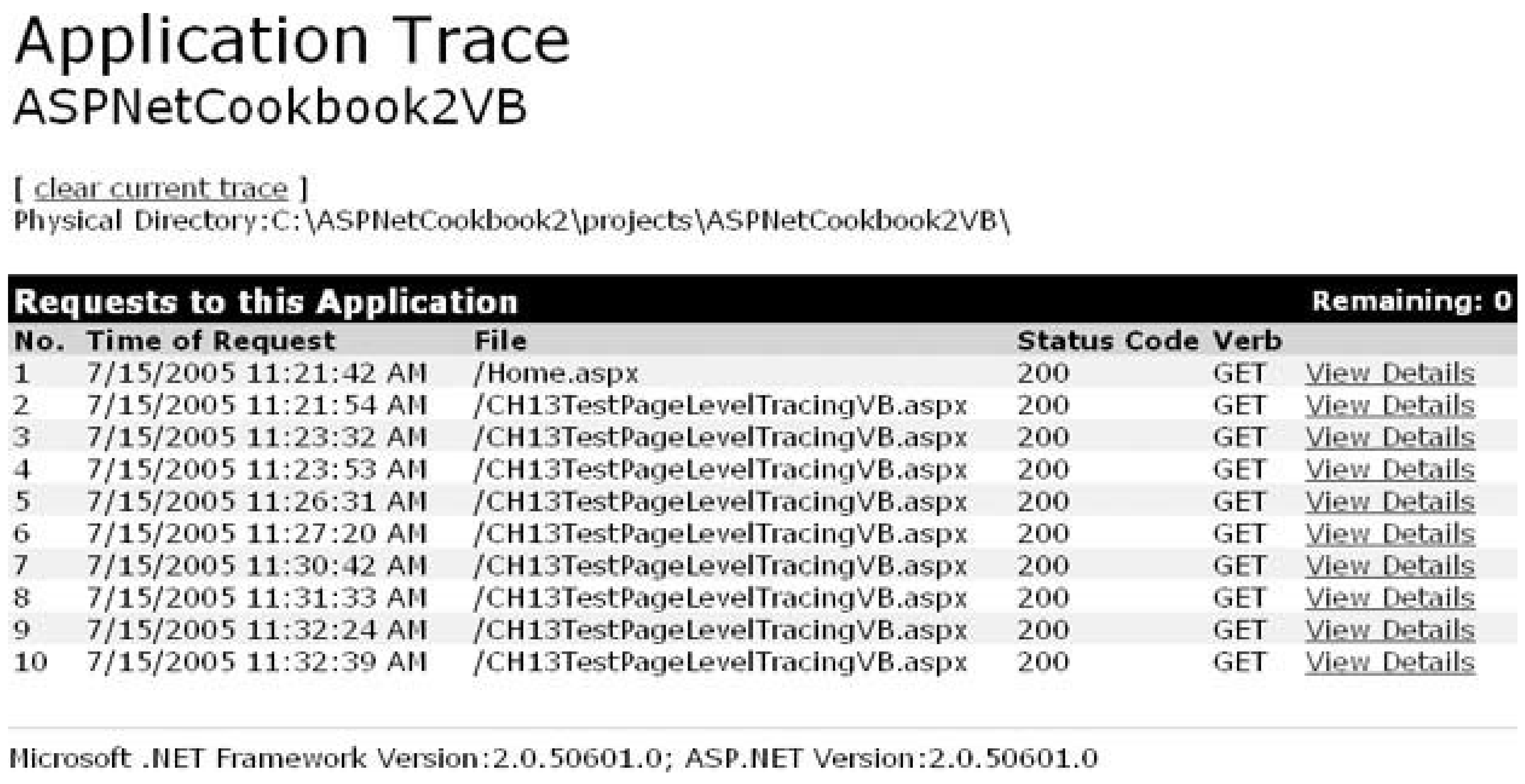
```
<configuration>
 <system.web>
  <trace enabled="true" />

 </system.web>
</configuration>
```

3. View the application trace log by browsing to the *trace.axd* page from the application root, like this:

   http://localhost/<yourapplicationname>/trace.axd

Figure 13-2 shows some sample trace log output.

Figure 13-2. Application-level tracing output (trace.axd)



Application Trace
ASPNetCookbook2VB

[clear current trace]
Physical Directory:C:\ASPNetCookbook2\projects\ASPNetCookbook2VB\

Requests to this Application					Remaining: 0
No.	Time of Request	File	Status Code	Verb	
1	7/15/2005 11:21:42 AM	/Home.aspx	200	GET	View Details
2	7/15/2005 11:21:54 AM	/CH13TestPageLevelTracingVB.aspx	200	GET	View Details
3	7/15/2005 11:23:32 AM	/CH13TestPageLevelTracingVB.aspx	200	GET	View Details
4	7/15/2005 11:23:53 AM	/CH13TestPageLevelTracingVB.aspx	200	GET	View Details
5	7/15/2005 11:26:31 AM	/CH13TestPageLevelTracingVB.aspx	200	GET	View Details
6	7/15/2005 11:27:20 AM	/CH13TestPageLevelTracingVB.aspx	200	GET	View Details
7	7/15/2005 11:30:42 AM	/CH13TestPageLevelTracingVB.aspx	200	GET	View Details
8	7/15/2005 11:31:33 AM	/CH13TestPageLevelTracingVB.aspx	200	GET	View Details
9	7/15/2005 11:32:24 AM	/CH13TestPageLevelTracingVB.aspx	200	GET	View Details
10	7/15/2005 11:32:39 AM	/CH13TestPageLevelTracingVB.aspx	200	GET	View Details

Microsoft .NET Framework Version:2.0.50601.0; ASP.NET Version:2.0.50601.0

Discussion

By adding a `<trace>` element to *web.config* and setting its enabled attribute to `"TRue"`, you can activate application-level tracing.

```
<trace enabled="true" />
```

What happens is that ASP.NET collects trace information for each HTTP request to the application anc directs it to the application trace log. You can view the application trace log in the trace viewer. To view the trace viewer, request *trace.axd* from the root of your application directory:

http://localhost/<yourapplicationname>/trace.axd

> The *trace.axd* is not a page but rather a special URL intercepted by ASP.NET. The *trace.axd* is an HTTP handler, the equivalent of an ISAPI extension. Chapter 20 provides recipes on how to create your own HTTP handlers.

The *trace.axd* shows you a sequential listing of the HTTP requests processed by your application, as shown in Figure 13-2.

Here are some of the more commonly used `<trace>` element attributes:

`requestLimit`

The default number of HTTP requests stored in the application trace log is 10. You can increase the number of HTTP requests using the `requestLimit` attribute:

```
<trace enabled="true" requestLimit="40" />
```

Once the limit is reached, if the `mostRecent` attribute is set `false`, then no other HTTP requests will be logged until the application is restarted, or you hit the "clear current trace" link on the *trace.axd* page (see the upper-right corner of Figure 13-2), or the query string `clear=1` is passed to *trace.axd*, like so:

http://localhost/<yourapplicationname>/trace.axd?clear=1

`mostRecent`

The default value is `false`, which indicates that once the `requestLimit` is reached, no additional trace information is stored. You can set the value to `TRue` if you want to see the most recent requests.

`pageOutput`

If, in addition to viewing the *trace.axd* file, you want to see trace information displayed at the bottom of the page that it is associated with, add `pageOutput="true"` to the `<trace>` element:

```
<trace enabled="true" pageOutput="true" />
```

The trace information you will see is identical to what would appear had you placed `TRace="true"` in the `@ Page` directive for the page (see Recipe 13.1 for details).

`localOnly`

To show trace information to the local user (i.e., the browser making the request is on the machine serving the request) but not to remote users, ensure the `<trace>` element includes `localOnly="true"`:

```
<trace enabled="true" pageOutput="true" localOnly="true" />
```

If you are viewing the trace log in the trace viewer and you want to see specific information about a request, like the kind you see in Figure 13-1, click the "View Details" link to the right.

When deploying your application to a production environment, you can explicitly disable *trace.axd* by placing `<httpHandler>` elements like these in *web.config:*

```
<configuration>
  <system.web>
    <httpHandlers>
      <remove verb="*" path="trace.axd" />
    </httpHandlers>
  </system.web>
</configuration>
```

See Also

Recipe 13.1 and Chapter 20

PREV

Recipe 13.4. Pinpointing the Cause of an Exception

Problem

You want to identify problems only when an exception occurs.

Solution

Dynamically turn on page-level tracing from within the `Catch` block of your exception handler and write to the trace log.

In the code-behind class for the page, use the .NET language of your choice to:

1. Set `Page.Trace.IsEnabled = true` in the `Catch` block of your exception handler.
2. Write to the trace log by using a `TRace.Write` of the form `TRace.Write("Exception"`, *`"Message", exc`*).

Figure 13-3 shows the appearance of some exception information in the trace sequence. Examples 13-4, 13-5 through 13-6 show the *.aspx* file and VB and C# code-behind files for the application that produces this result.

Discussion

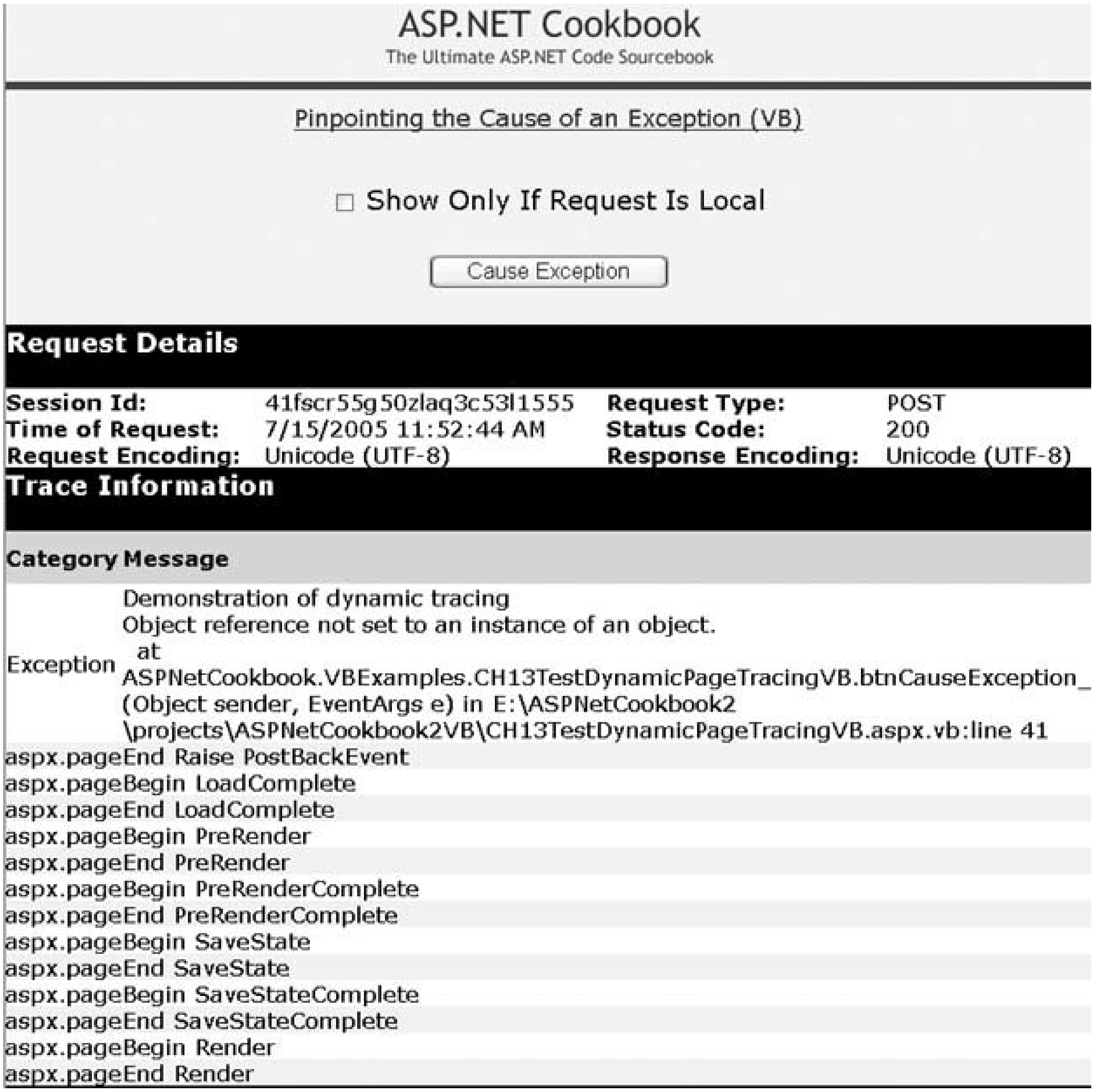
ASP.NET processes and displays trace statements only when tracing is enabled. However, what if you don't want to see the trace log all the time but only when an exception occurs? The answer is to turn tracing on dynamically for the page. You can then write the exception information to the trace log and debug the problem from there.

Our example that illustrates this solution is primitive, in that it forces an exception. Though this is no something you would normally do in production code, it does allow us to show the infrastructure needed to control tracing at runtime.

When the exception occurs, the exception handler enables the trace output by setting `TRace.IsEnabled` to `true`. For the exception information to appear in the trace sequence, you must use a `TRace.Write` of the form `trace.Write("Exception"`, *`"Message", exc`*) where *`exc`* is the `Exception` object defined in the `catch` statement.

Additionally, the code limits who sees the trace sequence, something you might want to consider if you are loathe to show tracing information to remote users when an exception occurs. Before activating tracing, the program checks to see whether the application is being run from the local machine (i.e., the browser making the request is on the machine serving the request). It does so by using the `Request` object to get the local IP address from the server variables. It compares this local address to the *loopback address*, a special IP number (127.0.0.1) that is designated for the software loopback interface of a machine. If it is not equal to the loopback address, a further comparison will be made to see if the local IP address from the server variables is the same as the local IP address that accompanied the request. The results of the comparison are used to determine whether to enable tracing and display the exception information in the trace sequence.

Figure 13-3. Exception information in the trace sequence


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Pinpointing the Cause of an Exception (VB)
Show Only If Request Is Local
Cause Exception

Request Details

Session Id:	41fscr55g50zlaq3c53l1555	**Request Type:**	POST
Time of Request:	7/15/2005 11:52:44 AM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message
Exception	Demonstration of dynamic tracing Object reference not set to an instance of an object. at ASPNetCookbook.VBExamples.CH13TestDynamicPageTracingVB.btnCauseException_(Object sender, EventArgs e) in E:\ASPNetCookbook2\projects\ASPNetCookbook2VB\CH13TestDynamicPageTracingVB.aspx.vb:line 41
aspx.page	End Raise PostBackEvent
aspx.page	Begin LoadComplete
aspx.page	End LoadComplete
aspx.page	Begin PreRender
aspx.page	End PreRender
aspx.page	Begin PreRenderComplete
aspx.page	End PreRenderComplete
aspx.page	Begin SaveState
aspx.page	End SaveState
aspx.page	Begin SaveStateComplete
aspx.page	End SaveStateComplete
aspx.page	Begin Render
aspx.page	End Render

The URLs for page requests on the local machine can be in the form *http://localhost/site/page* or *http://<server>/site/page* (where *<server>* is the local server name), so it is necessary to check for the loopback IP address as well as the IP address. If the page were requested using the server name in the URL, it would not be detected as a local request if the second check were not performed.

See Also

Search `IPAddress.IsLoopback` method in the MSDN library for another similar approach.

Example 13-4. Pinpointing the cause of an exception (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH13TestDynamicPageTracingVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH13TestDynamicPageTracingVB"
 Title="Test Dynamic Page Tracing" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Pinpointing the Cause of an Exception (VB)
 </div>
 <table width="90%" align="center" border="0">
  <tr>
   <td align="center" class="LabelText">
    <br />
    <input id="chkOnlyLocal" runat="server" type="checkbox" />
    Show Only If Request Is Local
   </td>
  </tr>
  <tr>
   <td align="center">
    <br />
    <input id="btnCauseException" runat="server"
    type="button"
    value="Cause Exception"
    onserverclick="btnCauseException_ServerClick" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 13-5. Pinpointing the cause of an exception code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH13TestDynamicPageTracingVB.aspx
 ''' </summary>
 Partial  Class  CH13TestDynamicPageTracingVB
  Inherits System.Web.UI.Page
  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the cause exception
  ''' button click event. It is responsible for causing an exception
  ''' to demonstrate dynamic tracing

  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnCauseException_ServerClick(ByVal sender As Object, _
          ByVal e As System.EventArgs)
   Dim list As ArrayList

  Try
   'force an exception by accessing the list without creating it first
   list.Add(0)

  Catch exc As Exception
   'enable tracing and output the exception information
   If ((Not chkOnlyLocal.Checked) OrElse _
    ((chkOnlyLocal.Checked) And (requestIsFromLocalMachine( )))) Then
   Trace.IsEnabled = True
   Trace.Write("Exception", _
    "Demonstration of dynamic tracing", _
    exc)
    End If
  End Try
 End Sub 'btnCauseException_ServerClick

 '''*************************************************************************
 ''' <summary>
 ''' This routine checks to see if the page request came from the local
 ''' machine.
 ''' </summary>
 '''
 ''' <returns>True if the request is from the local machine. Else, False
 ''' </returns>
 '''
 ''' <remarks>
 ''' NOTE: Since requests on a local machine can be in the form
 '''  http://localhost/site/page or http://server/site/page,
 '''  two checks are required. The first is for the localhost
 '''  loopback IP address (127.0.0.1) and the second is for the actual
 '''  IP address of the requestor.
 ''' </remarks>
 Private Function requestIsFromLocalMachine( ) As Boolean
  Dim isLocal As Boolean
  Dim localAddress As String

  ' Is browser fielding request from localhost?
  isLocal = Request.UserHostAddress.Equals("127.0.0.1")
  If (Not isLocal) Then
   ' Get local IP address from server variables
   localAddress = Request.ServerVariables.Get("LOCAL_ADDR")

   ' Compare local IP with IP address that accompanied request
   isLocal = Request.UserHostAddress.Equals(localAddress)
  End If

   Return (isLocal)
  End Function 'IsRequestFromLocalMachine
 End Class 'CH13TestDynamicPageTracingVB
End Namespace
```

Example 13-6. Pinpointing the cause of an exception code-behind (.cs)

```
using System;
using System.Collections;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH13TestDynamicPageTracingCS.aspx
 /// </summary>
 public partial class CH13TestDynamicPageTracingCS : System.Web.UI.Page
 {

   ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the cause exception
  /// button click event. It is responsible for causing an exception
  /// to demonstrate dynamic tracing
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnCauseException_ServerClick(object sender,
          System.EventArgs e)
  {
   ArrayList list = null;

   try
   {
    // force an exception by accessing the list without creating it first
     list.Add(0);
   } // try

   catch (Exception exc)
   {
    // enable tracing and output the exception information
    if ((!chkOnlyLocal.Checked) ||
    ((chkOnlyLocal.Checked) &&(requestIsFromLocalMachine( ))))
    {
    Trace.IsEnabled = true;
    Trace.Write("Exception",
     "Demonstration of dynamic tracing",
     exc);
   }
  } // catch
 } // btnCauseException_ServerClick

 ///*************************************************************************
  /// <summary>
  /// This routine checks to see if the page request came from the local
  /// machine.
  /// </summary>
  ///
  /// <returns>True if the request is from the local machine. Else, False
  /// </returns>
  ///
  /// <remarks>
  /// NOTE: Since requests on a local machine can be in the form
  ///  http://localhost/site/page or http://server/site/page,
  ///  two checks are required. The first is for the localhost
  ///  loopback IP address (127.0.0.1) and the second is for the actual
  ///  IP address of the requestor.
  /// </remarks>
  private Boolean requestIsFromLocalMachine( )
  {
   Boolean isLocal;
   string localAddress;

   // Is browser fielding request from localhost?
```

```
    isLocal = Request.UserHostAddress.Equals("127.0.0.1");
    if (!isLocal)
    {
     // Get local IP address from server variables
     localAddress = Request.ServerVariables.Get("LOCAL_ADDR");
     // Compare local IP with IP address that accompanied request
     isLocal = Request.UserHostAddress.Equals(localAddress);
    }

    return (isLocal);
   } // IsRequestFromLocalMachine
  }  //  CH13TestDynamicPageTracingCS
 }
```

PREV

Recipe 13.5. Uncovering Problems Within Web Application Components

Problem

You want to identify problems within a component of your web application, but your attempts to do so don't seem to work. When you make a call to `Trace.Write` in the business object, you get a compilation error or the debugger jumps right over the call and no output appears in the trace sequence.

Solution

Import the `System.Web` namespace and reference the current HTTP context when performing a `trace.Write` from within the component.

In the component class, use the .NET language of your choice to:

1. Import the `System.Web` namespace.
2. Reference the current HTTP context when performing a `TRace.Write`,asin `HTTPContext.Current.Trace.Write`.

The sample component we've written to illustrate this solution appears in Examples 13-7 (VB) and 13-8 (C#). Example 13-9 shows the *.aspx* file used to test the sample component. The code-behind for the test page appears in Examples 13-10 (VB) and 13-11 (C#). Figure 13-4 shows some sample output, including the resulting trace sequence.

Discussion

For `trace.Write` to work from within a component, you must be able to access the context for the current HTTP request. The easiest way to accomplish this is to import the `System.Web` namespace and access the `HTTPContext.Current` property from within the component.

If a component is not part of your web application, you will need to add a reference to the `System.Web.dll` assembly in your project. You do this in Visual Studio by selecting the project containing the component in the Solution Explorer. Right-click, and then select Add Reference. In the .NET tab of the displayed dialog box, select System.Web.dll from the list ofcomponents, click Select, and click OK.

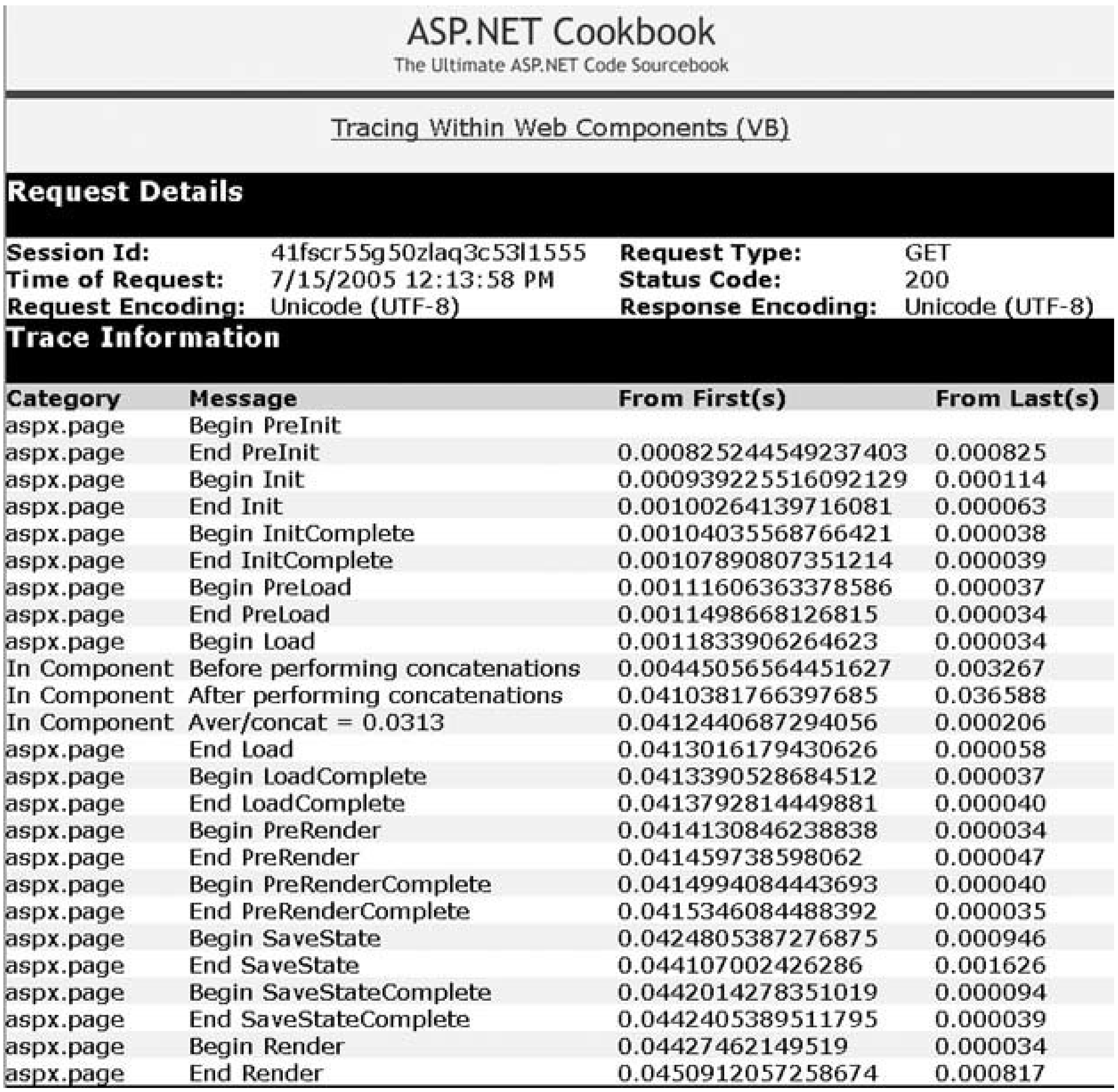
The HTTPContext provides access to the trace object that allows your application to write trace information, as shown in Examples 13-7 (VB) and 13-8 (C#).

> There is one major caveat to this sample: the disadvantage of referencing the current HTTP context in your component is that it does not allow the component to be used in non-web applications. If you need to share components in web and non-web applications, you may want to consider creating a Listener subclass instead, as described in Recipe 13.5.

See Also

Recipe 13.5

Figure 13-4. Trace sequence from testing the component



ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Tracing Within Web Components (VB)

Request Details

Session Id:	41fscr55g50zlaq3c53l1555	**Request Type:**	GET
Time of Request:	7/15/2005 12:13:58 PM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.000825244549237403	0.000825
aspx.page	Begin Init	0.000939225516092129	0.000114
aspx.page	End Init	0.00100264139716081	0.000063
aspx.page	Begin InitComplete	0.00104035568766421	0.000038
aspx.page	End InitComplete	0.00107890807351214	0.000039
aspx.page	Begin PreLoad	0.00111606363378586	0.000037
aspx.page	End PreLoad	0.0011498668126815	0.000034
aspx.page	Begin Load	0.0011833906264623	0.000034
In Component	Before performing concatenations	0.00445056564451627	0.003267
In Component	After performing concatenations	0.0410381766397685	0.036588
In Component	Aver/concat = 0.0313	0.0412440687294056	0.000206
aspx.page	End Load	0.0413016179430626	0.000058
aspx.page	Begin LoadComplete	0.0413390528684512	0.000037
aspx.page	End LoadComplete	0.0413792814449881	0.000040
aspx.page	Begin PreRender	0.0414130846238838	0.000034
aspx.page	End PreRender	0.041459738598062	0.000047
aspx.page	Begin PreRenderComplete	0.0414994084443693	0.000040
aspx.page	End PreRenderComplete	0.0415346084488392	0.000035
aspx.page	Begin SaveState	0.0424805387276875	0.000946
aspx.page	End SaveState	0.044107002426286	0.001626
aspx.page	Begin SaveStateComplete	0.0442014278351019	0.000094
aspx.page	End SaveStateComplete	0.0442405389511795	0.000039
aspx.page	Begin Render	0.04427462149519	0.000034
aspx.page	End Render	0.0450912057258674	0.000817

Example 13-7. The business service class (.vb)

```
Option Explicit On
Option Strict On

Imports System

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides a "web" component for demonstrating outputting
 ''' trace information from within a class
 ''' </summary>
 Public  Class  CH13TestWebComponentVB

  Private mStr As String

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the string in the object
  ''' </summary>
  Public Property theString( ) As String
   Get
    Return (mStr)
   End Get
   Set(ByVal Value As String)
    mStr = Value
   End Set
  End Property 'theString

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the ability to add the passed string to the
  ''' private string in this object one or more times.
  ''' </summary>
  '''
  ''' <param name="stringToAdd">Set to the string that is part of this object
  ''' </param>
  ''' <param name="numberOfCopies">Set to the number of copies to add
  ''' </param>
  Public Sub addToString(ByVal stringToAdd As String, _
     ByVal numberOfCopies As Integer)
   Dim counter As Integer
   Dim startTime As DateTime
   Dim elapsedTime As TimeSpan
   Dim averageTime As Double

   'output trace message indicating the start of the concatenations
   HttpContext.Current.Trace.Write("In Component", _
        "Before performing concatenations")

   'concatenation the passed string as requested
   startTime = DateTime.Now( )
   For counter = 1 To numberOfCopies
    mStr &= stringToAdd
   Next

   'output trace message indicating the end of the concatenations
   HttpContext.Current.Trace.Write("In Component", _
        "After performing concatenations")

   'calculate the elapsed time for the string concatenations
   elapsedTime = DateTime.Now.Subtract(startTime)
```

```
    'Write average time per concatenation in milliseconds to trace sequence
    averageTime = elapsedTime.TotalMilliseconds / numberOfCopies
    HttpContext.Current.Trace.Write("In Component", _
       "Aver/concat = " &averageTime.ToString("0.0000"))
   End Sub 'addToString

   '''*************************************************************************
   ''' <summary>
   ''' This constructor creates the object and initializes the variables
   ''' in the object
   ''' </summary>
   Public Sub New( )
    'initialize string in object
    mStr = ""
   End Sub 'New
  End Class 'CH13TestWebComponentVB
End Namespace
```

Example 13-8. The business service class (.cs)

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web;
using System.Web.Caching;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides a "web" component for demonstrating outputting
 /// trace information from within a class
 /// </summary>
 public class CH13TestWebComponentCS
 {
  private String mStr;

   ///*************************************************************************
  /// <summary>
  /// This property provides the ability to get/set the string in the object
  /// </summary>
  public String theString
  {
   get
   {
    return(mStr);
   }
   set
   {
    mStr = value;
   }
  } // theString

  ///*************************************************************************
  /// <summary>
  /// This routine provides the ability to add the passed string to the
  /// private string in this object one or more times.
  /// </summary>
  ///

  /// <param name="stringToAdd">Set to the string that is part of this object
  /// </param>
  /// <param name="numberOfCopies">Set to the number of copies to add
  /// </param>
  public void addToString(String stringToAdd,
     int numberOfCopies)
  {
   int counter;
   DateTime startTime;
   TimeSpan elapsedTime;
   Double averageTime;

   // output trace message indicating the start of the concatenations
   HttpContext.Current.Trace.Write("In Component",
        "Before performing concatenations");

   // concatenation the passed string as requested
   startTime = DateTime.Now;
   for (counter = 1; counter <numberOfCopies; counter++)
   {
    mStr += stringToAdd;
   }

   // output trace message indicating the end of the concatenations
   HttpContext.Current.Trace.Write("In Component",
        "After performing concatenations");

   // calculate the elapsed time for the string concatenations
   elapsedTime = DateTime.Now.Subtract(startTime);

   // Write average time per concatenation in milliseconds to trace sequence
   averageTime = elapsedTime.TotalMilliseconds / numberOfCopies;
   HttpContext.Current.Trace.Write("In Component",
     "Aver/concat = " + averageTime.ToString("0.0000"));

   } // addToString

  ///*************************************************************************
  /// <summary>
  /// This constructor creates the object and initializes the variables
```

```
  /// in the object
  /// </summary>
  public CH13TestWebComponentCS( )
  {
   // initialize string in object
   mStr = "";
  } // CH13TestWebComponentCS
 } // CH13TestWebComponentCS
}
```

Example 13-9. Code to test tracing in the component (.aspx)

```
<%@ Page Trace="True" Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH13TestTraceWithinWebComponentVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH13TestTraceWithinWebComponentVB"
 Title="Test Trace Within Web Component" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Tracing Within Web Components (VB)
 </div>
</asp:Content>
```

Example 13-10. Code to test tracing in the component code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH13TestTraceWithinWebComponentVB.aspx
 ''' </summary>
 Partial Class CH13TestTraceWithinWebComponentVB
  Inherits System.Web.UI.Page
  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
   Dim webComponent As CH13TestWebComponentVB

   'create the "web aware" component
   webComponent = New CH13TestWebComponentVB

   'add a string to the string in the component 1000 times
   webComponent.addToString("1234567890", _
      1000)

  End Sub 'Page_Load
 End  Class  'CH13TestTraceWithinWebComponentVB
End Namespace
```

Example 13-11. Code to test tracing in the component code-behind (.cs)

```
using System;
using System.Configuration;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH13TestTraceWithinWebComponentCS.aspx
 /// </summary>
 public partial class CH13TestTraceWithinWebComponentCS : System.Web.UI.Page
 {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   CH13TestWebComponentCS webComponent = null;

   // create the "web aware" component
   webComponent = new CH13TestWebComponentCS( );

   // add a string to the string in the component 1000 times
   webComponent.addToString("1234567890",
              1000);
    } // Page_Load
  }  //  CH13TestTraceWithinWebComponentCS
}
```

PREV

Recipe 13.6. Uncovering Problems Within Dual-Use Components

Problem

Because you intend to use a business component in web and non-webapplications, you want to enable tracing within the component without having to reference its current HTTP context.

Solution

Modify *web.config*, as shown in Example 13-12, to add the `WebPageTraceListener` listener to the `Listeners` collection and make it available to your application.

In your non-web-specific components, add plain-vanilla `TRace.Write` statements to output any desired information to the trace log, as shown in our sample component inExamples 13-13 (VB) and 13-14 (C#).

A web form and the associated VB and C# code-behind we've written to test the tracing in our non-web-specific component are shown in Examples 13-15, 13-16 through 13-17.

Discussion

The .NET Framework uses the concept of *trace listeners* in its handling of trace messages. By default, the `TraceListeners` collection contains a single listener (`DefaultTraceListener`) when you enable tracing. Additional listeners can be added via the*web.config* file or programmatically. When a `TRace.Write` is executed, all listeners in the `TRaceListeners` collection receive and process the message. This mechanism allows you to add trace statements to yourcomponents without the need to add a reference to the `System.Web` assembly.

ASP.NET 1.x does not provide any functionality to write trace information within your application's business and data tiers and display it in the page trace data. If you need this functionality for your 1. application, you will have to write a custom trace listener. ASP.NET 2.0 provides the `WebPageTraceListener` class to support writing trace information from anywhere in your application and displaying it in the page trace information.

The custom `WebPageTraceListener` is available by adding the entry to *web.config*, shown in Example 13-12.

When adding a WebPageTraceListener to your *web.config* file, you must specify the type correctly. The type attribute must be specified as shown here:

```
type="namespace, assembly"
```

The namespace must be the fully qualified namespace of the WebPageTraceListener. The assembly must be the name of the assembly containing the WebPageTraceListener (System.Web).

Our example business service class is nearly identical to the one we used in Recipe 13.4. Here are th differences:

- The imports (or using) statement at the beginning of the class is changed fromSystem.Web to System.Diagnostics. (The System.Diagnostics namespace provides the abstract base class for the trace listeners.)
- The HttpContext.Current.Trace.Write statements are changed to trace.Write.

Our test web form, like our example business service class, is nearly identical to our test web form used in Recipe 13.4. The only difference is that it uses our example business service class.

There are two advantages to the approach this recipe takes over the previous recipe:

- You can use plain-vanilla TRace.Writes in your component; they don't have to reference the current HTTP context, thus maintaining the component's compatibility for non-web uses.
- You can turn tracing on and off via a configuration file.

All classes containing Trace statements must be compiled with tracing enabled. If tracing is not enabled, the Trace statements will not be compiled into the output and, thus, will not be executed in the application.

Trace statements can be enabled by explicitly adding theTRACE compiler directive to the top of the classes in which you want tracing enabled:

```
#CONST TRACE = true

#define TRACE
```

Alternately, tracing can be enabled for the entire application in*web.config*:

VB

```
 <configuration>

   …

  <system.codedom>
   <compilers>
    <compiler language="VB"
     extension=".vb"
     compilerOptions="/d:Trace=true"
     type="Microsoft.VisualBasic.VBCodeProvider,
       System,
       Version=2.0.0.0,
       Culture=neutral,
        PublicKeyToken=b77a5c561934e089" />
   </compilers>
  </system.codedom>
 </configuration>
```

C#

```
 <configuration>

  …

 <system.codedom>
  <compilers>
   <compiler language="c#;cs;csharp"
     extension=".cs"
     compilerOptions="/d:TRACE"
     type="Microsoft.CSharp.CSharpCodeProvider,
       System,
       Version=2.0.3500.0,
       Culture=neutral,
        PublicKeyToken=b77a5c561934e089" />


    </compilers>
    </system.codedom>

    …

    </configuration>
```

See Also

Recipe 13.4

Example 13-12. Web.config settings to add the trace listener

```
<trace autoflush="true" indentsize="0">
 <listeners>
  <add name="WebPageTraceListener"
   type="System.Web.WebPageTraceListener,
     System.Web,
      Version=2.0.3600.0,
     Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a" />
 </listeners>
</trace>
```

Example 13-13. Business service class with plain-vanilla Trace.Writes (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Diagnostics

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides a "non-web" component for demonstrating outputting
 ''' trace information from within a class
 ''' </summary>
 Public Class CH13TestNonWebComponentVB

  Private mStr As String

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the string in the object
  ''' </summary>
  Public Property theString( ) As String
   Get
    Return (mStr)
   End Get
   Set(ByVal Value As String)
    mStr = Value
   End Set
  End Property 'theString

  '''*************************************************************************
```

```
    ''' <summary>
    ''' This routine provides the ability to add the passed string to the
    ''' private string in this object one or more times.
    ''' </summary>
    '''
    ''' <param name="stringToAdd">Set to the string that is part of this object
    ''' </param>
    ''' <param name="numberOfCopies">Set to the number of copies to add
    ''' </param>
    Public Sub addToString(ByVal stringToAdd As String, _
       ByVal numberOfCopies As Integer)
     Dim counter As Integer
     Dim startTime As DateTime
     Dim elapsedTime As TimeSpan
     Dim averageTime As Double

     'output trace message indicating the start of the concatenations
     Trace.Write("In Non-web Component", _
      "Before performing concatenations")

     'concatenation the passed string as requested
     startTime = DateTime.Now( )
     For counter = 1 To numberOfCopies
      mStr &= stringToAdd
     Next

     'output trace message indicating the end of the concatenations
     Trace.Write("In Non-web Component", _
      "After performing concatenations")

     'calculate the elapsed time for the string concatenations
     elapsedTime = DateTime.Now.Subtract(startTime)

     'Write average time per concatenation in milliseconds to trace sequence
     averageTime = elapsedTime.TotalMilliseconds / numberOfCopies
     Trace.Write("In Non-web Component", _
      "Aver/concat = " &averageTime.ToString("0.0000"))
    End Sub 'addToString

    '''*************************************************************************
    ''' <summary>
    ''' This constructor creates the object and initializes the variables
    ''' in the object
    ''' </summary>
    Public Sub New( )
     'initialize string in object
     mStr = ""
    End Sub 'New
  End Class 'CH13TestNonWebComponentVB
End Namespace
```

Example 13-14. Business service class with plain-vanilla Trace.Writes (.cs

```
using System;
using System.Diagnostics;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides a "non-web" component for demonstrating outputting
 /// trace information from within a class
 /// </summary>
 public  class  CH13TestNonWebComponentCS
 {
  private String mStr;

   ///***************************************************************************
  /// <summary>
  /// This property provides the ability to get/set the string in the object
  /// </summary>
  public String theString
  {
   get
   {
    return(mStr);
   }
   set
   {
    mStr = value;
   }
  } // theString

   ///***************************************************************************
  /// <summary>
  /// This routine provides the ability to add the passed string to the
  /// private string in this object one or more times.
  /// </summary>
  ///
  /// <param name="stringToAdd">Set to the string that is part of this object
  /// </param>
  /// <param name="numberOfCopies">Set to the number of copies to add
  /// </param>
  public void addToString(String stringToAdd,
   int numberOfCopies)
  {
   int counter;
   DateTime startTime;
   TimeSpan elapsedTime;
   Double averageTime;
```

```
    // output trace message indicating the start of the concatenations
    Trace.Write("In Non-web Component",
     "Before performing concatenations");

    // concatenation the passed string as requested
    startTime = DateTime.Now;
    for (counter = 1; counter <numberOfCopies; counter++)
    {
     mStr += stringToAdd;
    }

    // output trace message indicating the end of the concatenations
    Trace.Write("In Non-web Component",
     "After performing concatenations");

    // calculate the elapsed time for the string concatenations
    elapsedTime = DateTime.Now.Subtract(startTime);

    // Write average time per concatenation in milliseconds to trace sequence
    averageTime = elapsedTime.TotalMilliseconds / numberOfCopies;
    Trace.Write("In Non-web Component",
     "Aver/concat = " + averageTime.ToString("0.0000"));
   } // addToString

   ///*************************************************************************
   /// <summary>
   /// This constructor creates the object and initializes the variables
   /// in the object
   /// </summary>
   public CH13TestNonWebComponentCS( )
   {
    // initialize string in object
    mStr = "";
   } // CH13TestNonWebComponentCS
  } // CH13TestNonWebComponentCS
}
```

Example 13-15. Code to test tracing in the non-web component (.aspx)

```
<%@ Page Trace="true" Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH13TestTraceWithinNonWebComponentVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH13TestTraceWithinNonWebComponentVB"
 Title="Test Trace Within NonWeb Component" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Tracing Within Non-Web Components (VB)
 </div>
</asp:Content>
```

Example 13-16. Code to test tracing in the non-web component code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH13TestTraceWithinNonWebComponentVB.aspx
 ''' </summary>
 Partial  Class  CH13TestTraceWithinNonWebComponentVB
  Inherits System.Web.UI.Page
  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim nonWebComponent As CH13TestNonWebComponentVB
   'create the "non-web aware" component
   nonWebComponent = New CH13TestNonWebComponentVB

   'add a string to the string in the component 1000 times
   nonWebComponent.addToString("1234567890", _
       1000)
  End Sub 'Page_Load
 End  Class  'CH13TestTraceWithinNonWebComponentVB
End Namespace
```

Example 13-17. Code to test tracing in the non-web component code-behind (.cs)

```
using System;
using System.Configuration;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH13TestTraceWithinNonWebComponentCS.aspx
 /// </summary>
 public partial class CH13TestTraceWithinNonWebComponentCS :
  System.Web.UI.Page
 {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)

  {
   CH13TestNonWebComponentCS nonwebComponent = null;
   // create the "non-web aware" component
   nonwebComponent = new CH13TestNonWebComponentCS();

   // add a string to the string in the component 1000 times
   nonWebComponent.addToString("1234567890",
       1000);
  } // Page_Load
 } // CH13TestTraceWithinNonWebComponentCS
}
```

PREV

Recipe 13.7. Writing Trace Data to the Event Log with Controllable Levels

Problem

You want your application to output trace information to the event log and, at the same time, control what level of information is output.

Solution

Modify *web.config* to:

1. Add the `EventLogTraceListener` listener to the `Listeners` collection and make it available to your application, as shown in Example 13-18.
2. Add the `TraceSwitch`, as shown in Example 13-18.

In the classes you want to output trace information, create a `TRaceSwitch` object using the name of the `traceSwitch` you added to the *web.config* file, and use the `WriteIf` and `WriteLineIf` methods of the `TRace` class to output the required messages, as we demonstrate in our sample application shown in Examples 13-19, 13-20 through 13-21.

Discussion

The technique we advocate for writing trace information to the event log involves adding the `EventLogTraceListener` to the listener collection in *web.config* to write the trace information to the event log. We find it useful to control the level of messages output to the event log, such as outputting only error messages or outputting error and warning messages. Controlling the level of messages that are output involves the use of switches (more about this in a minute).

As discussed in Recipe 13.5, you can add additional listeners to the `traceListeners` collection via the *web.config* file. When a `TRace.Write` or `trace.WriteLine` is executed, all listeners in the `TRaceListeners` collection receive and process their output.

The support that the .NET Framework provides for `TRaceListeners` is more powerful when coupled with switchesw. *Switches* provide the ability to control when trace information is sent to the `traceListeners` configured for your application.

Two switch types are provided in the .NET Framework: `BooleanSwitch` and `TRaceSwitch`. The `BooleanSwitch` class supports two states (on and off) that turn the trace output on and off. The traceSwitch class supports five levels (off, error, warning, info, and verbose) to provide the ability to output messages only for the configured levels.

You must first add the switch and listener information to your *web.config* file, as shown in Example 13-18. The switch data includes the name of the switch and the value for the switch. The switch name is the name used in your code to access the switch configuration. The value defines the message level to output, as shown in Table 13-1.

Table 13-1. Switch level values

Value	Meaning
0	Output no messages
1	Output only error messages
2	Output error and warning messages
3	Output error, warning, and informational messages
4	Output all messages

To output trace messages that use the switch information, you need to create a traceSwitch object passing the name of the switch and a general description of the switch. After creating the traceSwitch, you use it with the WriteIf and WriteLineIf methods of the trace class to output your messages. The first parameter of either method defines the level for which the message should be output. In other words, if you only want the message to be output when the switch is configured for "warnings," set the first parameter to the traceWarning property of the switch you created. The second parameter should be set to the message you want to output.

We are not outputting the trace information to the web form, as we have in other examples in this chapter, so it is unnecessary to add the TRace="true" statement to the @ Page directive in the *.aspx* page or to turn on application-level tracing in the *web.config* file.

The name used in the constructor of the TRaceSwitch must match the name of the switch in the *web.config* file. If you fail to use the exact name defined in the *web.config* file, you can wind up spending a great deal of time trying to determine why your messages are not being output as expected.

In a web application, referencing the TRace class without further qualifying the namespace will actually reference the System.Web.Trace class, which does not support the WriteIf and WriteLineIf methods. To access the TRace class in the System.Diagnostics namespace that provides the WriteIf and WriteLineIf methods, fully qualify the reference:

```
System.Diagnostics.Trace.WriteIf(level,
      Message)
```

See Also

Recipe 13.5

Example 13-18. web.config settings for adding the trace listener and trac
switch

```
<configuration>

  …

 <system.diagnostics>
  <switches>
   <!-- This switch controls messages written to the event log.
   To control the level of message written to the log set
   the value attribute as follows:
   "0" - output no messages
   "1" - output only error messages
   "2" - output error and warning messages
   "3" - output error, warning, and informational messages
   "4" - output all messages
  -->
  <add name="EventLogSwitch" value="1"/>
 </switches>

 <trace  autoflush="true"  indentsize="0">
  <listeners>
   <add name="EventLogTraceListener"
    type="System.Diagnostics.EventLogTraceListener,
     System,
     Version=2.0.0.0,
     Culture=neutral,
     PublicKeyToken=b77a5c561934e089"
    initializeData="Application" />
   </listeners>
  </trace>
 </system.diagnostics>
</configuration>
```

Example 13-19. Writing trace information as a function of trace level (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH13TestTracingWithLevelControlVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH13TestTracingWithLevelControlVB"

 Title="Test Tracing With Level Control" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Tracing With Output To Event Log with Trace Levels (VB)
 </div>
</asp:Content>
```

Example 13-20. Writing trace information as a function of trace level (.vb

```
Option Explicit On
Option Strict On

Imports System
Imports System.Diagnostics

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH13TestTracingWithLevelControlVB.aspx
 ''' </summary>
 Partial  Class  CH13TestTracingWithLevelControlVB
  Inherits System.Web.UI.Page
  '''**************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim generalTraceSwitch As TraceSwitch

   'create the trace switch
   generalTraceSwitch = New TraceSwitch("EventLogSwitch", _
          "Used throughout the application")

   'write trace data if error level is enabled
   System.Diagnostics.Trace.WriteIf(generalTraceSwitch.TraceError, _
         "This is an error message")

   'write trace data if warning level is enabled
   System.Diagnostics.Trace.WriteIf(generalTraceSwitch.TraceWarning, _
         "This is an warning message")

   'write trace data if info level is enabled
   System.Diagnostics.Trace.WriteIf(generalTraceSwitch.TraceInfo, _
         "This is an info message")

   'write trace data if verbose level is enabled
   System.Diagnostics.Trace.WriteIf(generalTraceSwitch.TraceVerbose, _
         "This is an verbose message")

  End Sub 'Page_Load
 End  Class  'CH13TestTracingWithLevelControlVB
End  Namespace
```

Example 13-21. Writing trace information as a function of trace level (.cs)

```
using System;
using System.Diagnostics;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH13TestTracingWithLevelControlCS.aspx
 /// </summary>
 public partial class CH13TestTracingWithLevelControlCS : System.Web.UI.Page
 {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   TraceSwitch generalTraceSwitch = null;

   // create the trace switch
   generalTraceSwitch = new TraceSwitch("EventLogSwitch",
          "Used throughout the application");

   // write trace data if error level is enabled
   System.Diagnostics.Trace.WriteIf(generalTraceSwitch.TraceError,
        "This is an error message");

   // write trace data if warning level is enabled
   System.Diagnostics.Trace.WriteIf(generalTraceSwitch.TraceWarning,
        "This is an warning message");

   // write trace data if info level is enabled
   System.Diagnostics.Trace.WriteIf(generalTraceSwitch.TraceInfo,
        "This is an info message");

   // write trace data if verbose level is enabled
   System.Diagnostics.Trace.WriteIf(generalTraceSwitch.TraceVerbose,
        "This is an verbose message");
  } // Page_Load
 }  //  CH13TestTracingWithLevelControlCS
}
```

Recipe 13.8. Sending Trace Data via Email with Controllable Levels

Problem

You want your application to email trace information and, at the same time, control what level of information is sent.

Solution

Create your own trace listener that inherits from the `traceListener` class and overrides the `Write` and `WriteLine` methods to email their output. A sample trace listener we've written to demonstrate this solution is shown in Example 13-22 (VB) and Example 13-23 (C#).

Next, modify your *web.config* file to add the custom `TRaceListener` and `TRaceSwitch`, as shown in Example 13-24 (VB) and Example 13-25 (C#).

In the classes you want to output trace information, create a `traceSwitch` object using the name of the `TRaceSwitch` you added to the *web.config* file, and then use the `WriteIf` and `WriteLineIf` methods of the `TRace` class to output the required messages, as described in Recipe 13.6.

Discussion

The technique we advocate for emailing trace information involves creating your own custom trace listener. The purpose of the custom trace listener is to override the `Write` and `WriteLine` methods such that the information normally written to the trace sequence is emailed instead, as shown in Examples 13-22 (VB) and 13-23 (C#). When building a trace listener for this purpose, we find it usefu to control the level of messages emailed, such as sending only error messages or sending error and warning messages. Controlling the level of messages that are output involves the use of switches, as described in Recipe 13.6.

To provide the ability to configure the parameters of the email sent, we have added configuration elements to the `<appSettings>` element in *web.config*, as shown below. These provide the ability to change the to `address, from address, subject`, and `email server` used to send the email, without having to recompile the code.

```
<configuration>

  …
 <appSettings>
  <add key="EmailServer" value="mail.adelphia.net" />
   <add key="CH13EmailListenerToAddress" value="msmith@smith.com" />
   <add key="CH13EmailListenerFromAddress" value="appMessage@myapp.com" />

   <add key="CH13EmailListererSubject" value="Application Message" />
 </appSettings>

  …

</configuration>
```

See Also

Recipes 13.5 and 13.6

Example 13-22. Custom TraceListener for sending email (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Diagnostics
Imports System.Net.Mail

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides a trace listener that sends all trace messages
 ''' via email.
 ''' </summary>
 Public Class CH13EmailListenerVB
  Inherits TraceListener

   '''*************************************************************************
  ''' <summary>
  ''' This routine writes the passed message to the event log
  ''' </summary>
  '''
  ''' <param name="message">Set to the message to write</param>
  Public Overloads Overrides Sub Write(ByVal message As String)
   sendEmail(message)
  End Sub 'Write

   '''*************************************************************************
  ''' <summary>
  ''' This routine writes the passed message to the event log
  ''' </summary>
  '''
  ''' <param name="category">Set to the category for the message</param>
  ''' <param name="message">Set to the message to write</param>
  Public Overloads Overrides Sub Write(ByVal category As String, _
           ByVal message As String)
   sendEmail(category &": " &message)
  End Sub 'Write

  '''*************************************************************************
  ''' <summary>
  ''' This routine writes the passed message to the event log with a CR/LF
  ''' </summary>
  '''

  ''' <param name="message">Set to the message to write</param>
  Public Overloads Overrides Sub WriteLine(ByVal message As String)
   sendEmail(message)
  End Sub 'WriteLine

  '''*************************************************************************
  ''' <summary>
  ''' This routine writes the passed message to the event log with a CR/LF
  ''' </summary>
  '''
  ''' <param name="category">Set to the category for the message</param>
  ''' <param name="message">Set to the message to write</param>
  Public Overloads Overrides Sub WriteLine(ByVal category As String, _
         ByVal message As String)
   sendEmail(category &": " &message)
  End Sub 'WriteLine

  '''*************************************************************************
  ''' <summary>
  ''' This routine sends the passed message to the email address(es)
  ''' defined in web.config
  ''' </summary>
  '''
  ''' <param name="message"></param>
  Private Sub sendEmail(ByVal message As String)
   Dim toAddress As String
   Dim fromAddress As String
   Dim subject As String
   Dim client As SmtpClient
   Dim emailServer As String

   'get the to/from addressses and subject from web.config
   toAddress = ConfigurationManager.AppSettings("CH13EmailListenerToAddress")
   fromAddress = ConfigurationManager.AppSettings("CH13EmailListenerFromAddress")
   subject = ConfigurationManager.AppSettings("CH13EmailListererSubject")

   'send the message
```

```
    emailServer = ConfigurationManager.AppSettings("EmailServer")
    client = New SmtpClient(emailServer)
    client.Send(fromAddress, _
     toAddress, _
     subject, _
     message)
  End Sub 'sendEmail
 End Class 'CH13EmailListenerVB
End Namespace
```

Example 13-23. Custom TraceListener for sending email (.cs)

```
using System;
using System.Configuration;
using System.Diagnostics;
using System.Net.Mail;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides a "non-web" component for demonstrating outputting
 /// trace information from within a class
 /// </summary>
 public class CH13EmailListenerCS : TraceListener
 {
   ///***********************************************************************
  /// <summary>
  /// This routine writes the passed message to the event log
  /// </summary>
  ///
  /// <param name="message">Set to the message to write</param>
  public override void Write(String message)
  {
   sendEmail(message);
  } // Write

   ///***********************************************************************
  /// <summary>
  /// This routine writes the passed message to the event log
  /// </summary>
  ///
  /// <param name="category">Set to the category for the message</param>
  /// <param name="message">Set to the message to write</param>
  public override void Write(String category,
          String message)
  {
   sendEmail(category + ": " + message);
```

```
} // Write

///***************************************************************************
/// <summary>
/// This routine writes the passed message to the event log with a CR/LF
/// </summary>
///
/// <param name="message">Set to the message to write</param>
public override void WriteLine(String message)
{
 sendEmail(message);
} // WriteLine

///***************************************************************************
/// <summary>
/// This routine writes the passed message to the event log with a CR/LF
/// </summary>
///
/// <param name="category">Set to the category for the message</param>
/// <param name="message">Set to the message to write</param>
public override void WriteLine(String category,
         String message)

{
 sendEmail(category + ": " + message);
} // WriteLine

///***************************************************************************
/// <summary>
/// This routine sends the passed message to the email address(es)
/// defined in web.config
/// </summary>
///
/// <param name="message"></param>
private void sendEmail(String message)
{
 String toAddress;
 String fromAddress;
 String subject;
 SmtpClient client;
 String emailServer;

 // get the to/from addressses and subject from web.config
 toAddress = ConfigurationManager.
  AppSettings["CH13EmailListenerToAddress"];
 fromAddress = ConfigurationManager.
  AppSettings["CH13EmailListenerFromAddress"];
 subject = ConfigurationManager.
  AppSettings["CH13EmailListererSubject"];

 // send the message
 emailServer = ConfigurationManager.AppSettings["EmailServer"];
```

```
    client = new SmtpClient(emailServer);
    client.Send(fromAddress,
     toAddress,
     subject,
     message);
   } // sendEmail
  } // CH13EmailListenerCS
 }
```

Example 13-24. web.config settings for adding the trace listener and trac
switch (.vb)

```
<configuration>
 <appSettings>
  <add key="EmailServer" value="mail.adelphia.net" />
  <add key="CH13EmailListenerToAddress" value=" msmith@smith.com" />
  <add key="CH13EmailListenerFromAddress" value="appMessage@myapp.com" />
  <add key="CH13EmailListererSubject" value="Application Message" />
 </appSettings>

  …

 <system.diagnostics>
  <switches>

  <!-- This switch controls messages written to the event log.
    To control the level of message written to the log set
    the value attribute as follows:
    "0" - output no messages
    "1" - output only error messages
    "2" - output error and warning messages
    "3" - output error, warning, and informational messages
    "4" - output all messages
  -->
  <add name="EventLogSwitch" value="1"/>
 </switches>

 <trace autoflush="true" indentsize="0">
  <listeners>
   <add name="CookbookEventLogListener"
     type="ASPNetCookbook.VBExamples.CH13EmailListenerVB, __code" />
   </listeners>
  </trace>
 </system.diagnostics>
</configuration>
```

Example 13-25. web.config settings for adding the trace listener and trac
switch (.cs)

```
<configuration>
 <appSettings>
  <add key="EmailServer" value="mail.adelphia.net" />
   <add key="CH13EmailListenerToAddress" value=" msmith@smith.com" />
   <add key="CH13EmailListenerFromAddress" value="appMessage@myapp.com" />
   <add key="CH13EmailListererSubject" value="Application Message" />
 </appSettings>

  …

 <system.diagnostics>
  <switches>
   <!-- This switch controls messages written to the event log.
     To control the level of message written to the log set
     the value attribute as follows:
     "0" - output no messages
     "1" - output only error messages
     "2" - output error and warning messages
      "3" - output error, warning, and informational messages
     "4" - output all messages
   -->
    <add name="EventLogSwitch" value="1"/>
  </switches>

   <trace autoflush="true" indentsize="0">
    <listeners>
     <add name="CookbookEventLogListener"
      type="ASPNetCookbook.CSExamples.CH13EmailListenerCS, __code" />
     </listeners>

  </trace>
 </system.diagnostics>
</configuration>
```

Recipe 13.9. Using a Breakpoint to Stop Execution of an Application When a Condition Is Met

Problem

You have some fairly complicated code that is having a problem after many iterations, and you need an easy way to stop execution when the conditions are met.

Solution

Set a conditional breakpoint using an expression in the Visual Studio debugger. The value of the expression will determine whether program execution breaks when the breakpoint is hit.

To set a conditional breakpoint in the Visual Studio debugger:

1. Set a breakpoint in the usual fashion by clicking in the gray border to the left of the line where you want execution to break.
2. Right-click the breakpoint and select the Condition command from the menu.
3. Use the Conditions dialog box to set the conditions for the break, as shown in Figure 13-5 Typically, you'll set a conditional expression like any of these:

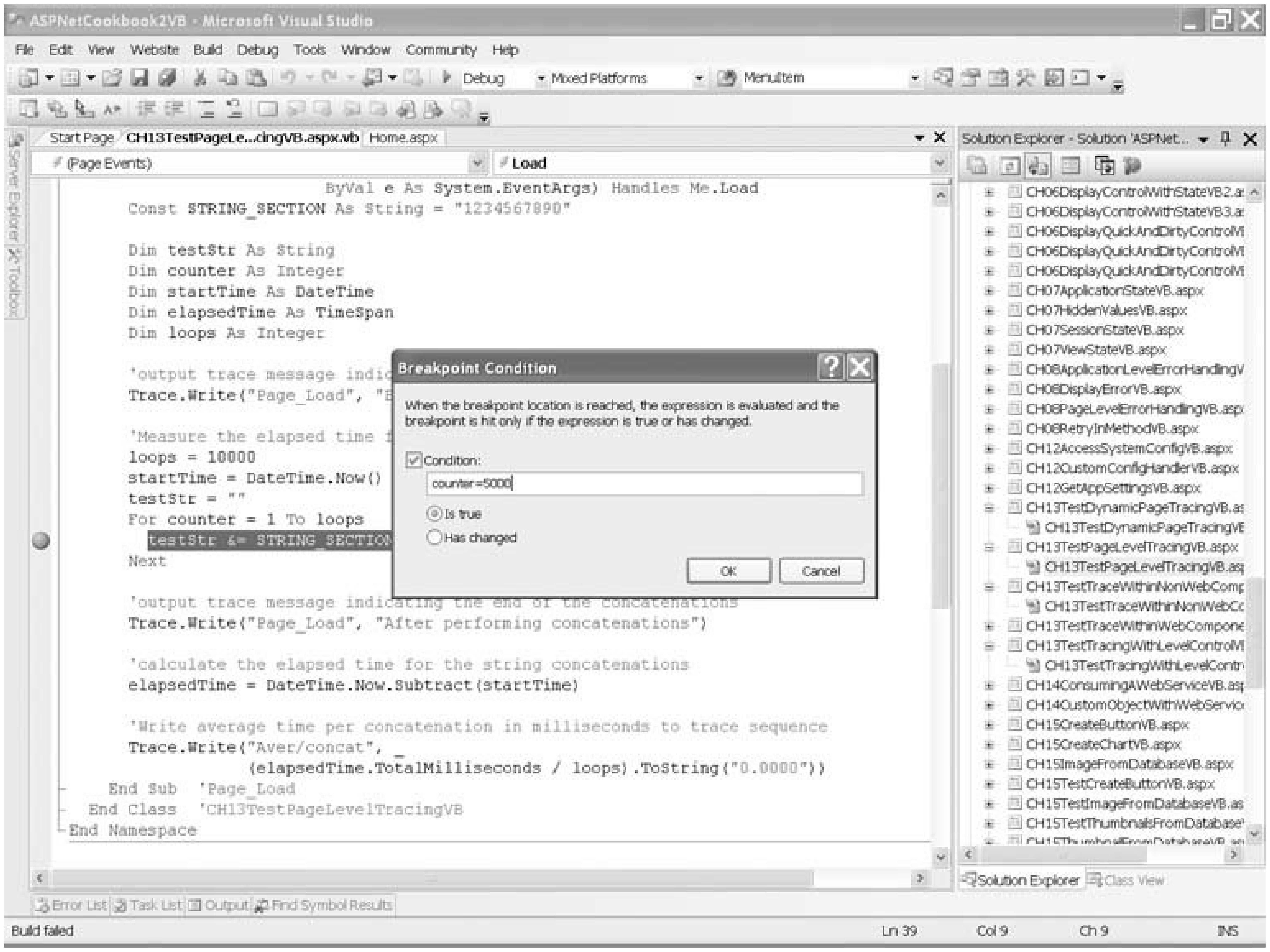
```
counter = 5000

i=100 AND j=150

message.Length > 0

counter == 5000

i==100 && j==150

message.Length > 0
```

When you run the program, execution will break at the location when the expression is true or has changed, depending on the option you've chosen in the dialog box.

Discussion

You can view the contents of the Visual Studio Locals window to verify the values of the variables involved in the expression. Access the Locals window by selecting the Debug Windows Locals command; the Locals window is accessible only when the Visual Studio Debugger is active.

Figure 13-5. Setting a conditional breakpoint in Visual Studio



```
                        ByVal e As System.EventArgs) Handles Me.Load
    Const STRING_SECTION As String = "1234567890"

    Dim testStr As String
    Dim counter As Integer
    Dim startTime As DateTime
    Dim elapsedTime As TimeSpan
    Dim loops As Integer

    'output trace message indic
    Trace.Write("Page_Load", "E

    'Measure the elapsed time f
    loops = 10000
    startTime = DateTime.Now()
    testStr = ""
    For counter = 1 To loops
      testStr &= STRING_SECTION
    Next

    'output trace message indicating the end of the concatenations
    Trace.Write("Page_Load", "After performing concatenations")

    'calculate the elapsed time for the string concatenations
    elapsedTime = DateTime.Now.Subtract(startTime)

    'Write average time per concatenation in milliseconds to trace sequence
    Trace.Write("Aver/concat", _
                (elapsedTime.TotalMilliseconds / loops).ToString("0.0000"))
  End Sub  'Page_Load
 End Class  'CH13TestPageLevelTracingVB
End Namespace
```

Breakpoint Condition

When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.

☑ Condition: counter=5000

[✓ Is true] Has changed

ASPNetCookbook2VB - Microsoft Visual Studio
File Edit View Website Build Debug Tools Window Community Help
Debug
Mixed Platforms
MenuItem
Start Page
CH13TestPageLe...cingVB.aspx.vb
Home.aspx
(Page Events)
Load
OK
Cancel
Solution Explorer - Solution 'ASPNet...
Solution Explorer
Class View
Error List
Task List
Output
Find Symbol Results
Build failed
Ln 39
Col 9
Ch 9
INS

Another approach is to set a hit count by selecting the Hit Count command from the menu. The hit count lets you specify how many times the breakpoint is hit before the debugger enters break mode. For example, you might choose to break when the hit count is equal to 100. The debugger will break only when the hit count reaches the target number.

See Also

All the rules for setting breakpoint expressions are available from Visual Studio Help under the "Expressions in the Debugger" topic, which is accessible from the Breakpoint Properties dialog box (the expression evaluator accepts most expressions written in Visual Basic or C#).

PREV

PREV

Chapter 14. Web Services

PREV

PREV

14.0 Introduction

Web services hold the promise of revolutionizing the way that organizations of all kinds share data. They provide a standard, universal means of data exchange within the grasp of the average programmer. And beyond that, they are easy to get started with.

For those new to the subject, web services are modular applications that can be described, published located, and invoked over standard Internet protocols using standardized XML messaging. Applications use the XML-based SOAP for the exchange of information in a loosely coupled, distributed environment. Applications posted to the Web are described with theWeb Services Description Language (WSDL) and are registered with a private or public service registry using the UDDI standard, such as http://uddi.microsoft.com or http://uddi.ibm.com.

This chapter shows you how to deal with a number of common web service scenarios and overcome some of the typical problems you might encounter. Detailed coverage of web services would require an entire book. For a tutorial on web services, we recommend*Programming .NET Web Services*, by Alex Ferrara and Matthew Mac-Donald (O'Reilly).

Web Service Enhancements (WSE) 2.0 is an add-on for Visual Studio that simplifies the development and deployment of secure web services. For more information on WSE, see http://msdn.microsoft.com/webservices/webservices/building/wse/default.aspx.

Recipe 14.2. Creating a Web Service

Problem

You want to create your own web service.

Solution

Use Visual Studio 2005 to create a new web service and then add methods to expose the functionalit required for your web service.

To create the new web service:

1. On the Visual Studio 2005 File menu, choose New Web Site....
2. When the New Web Site dialog box is displayed, select the ASP.NET Web Site template, the desired language, and the location for the web site.
3. Right-click on the new web site in the Solution Explorer, select Add New Item, and select the Web Service template.
4. Enter the name of the web service in the Name text box, select the desired language, and click Add.

Visual Studio 2005 creates a folder for the web site that contains an*asmx* file and code-behind files for the web service, like those shown in Examples 14-2, 14-3 , 14-4 through 14-5 . The web service is fully functional at this point but is a shell with no useful functionality.

Visual Studio 2005 places the code-behind files for web services in the *App_Code* directory instead of placing them in the same folder as the*.asmx* file, as Visual Studio 2003 did.

You need to add methods to the code-behind to expose the functionality required for your web service. The code-behind shown in Examples 14-6 (VB) and 14-7 (C#) shows a method we have added to our example web service to return a list of books from a database.

You can add a web service to an existing web application by selecting the web site in the Solution Explorer and then selecting Add New Item from the Website menu. When the Add New Item dialog box is displayed, select the Web Service template, enter the desired name of the web service, enter the desired language, and click Add. Any number of web services can be added to a web application.

Discussion

Web services are a useful tool for communicating with remote systems or with systems built with technologies different from those used to build your application.

Web services are convenient for wrapping legacy COM components. This is especially true when you need to access the functionality of those components from a different domain than where the components reside and the domains do not have a trust relationship.

Visual Studio 2005 simplifies the creation of web services. With a few menu selections, you can quickly build a web service shell that you can use to create the functionality required by your application.

In ASP.NET, a web service consists of an *.asmx* file and a code-behind class that provides the required functionality. The content of an *.asmx* file consists of a single line that contains a `WebService` directive. The line is like the `@ Page` directive used in the *.aspx* file but with `Page` replaced by `WebService` :

```
<%@ WebService Language="VB"
   CodeBehind="~/App_Code/CH14QuickWebServiceVB1.vb"
  Class="VBWebServices.CH14QuickWebServiceVB1"  %>


<%@ WebService Language="C#"
   CodeBehind="~/App_Code/CH14QuickWebServiceCS1.cs"
  Class="CSWebServices.CH14QuickWebServiceCS1"  %>
```

The code-behind file for a web service consists of a class that inherits from `System.Web.Services.WebService` .

```
Public  Class  CH14QuickWebServiceVB1
  Inherits System.Web.Services.WebService
  …
```

```
End Class 'CH14QuickWebServiceVB1
```

```
public class CH14QuickWebServiceCS1 : System.Web.Services.WebService
{
 …
} // CH14QuickWebServiceCS1
```

In addition, Visual Studio 2005 adds a `WebService` attribute to the class definition. Though not explicitly required, the `WebService` attribute lets you define the namespace for the web service. By default, the namespace is set to http://tempuri.org/, but you will typically want to set this to the URI representing your company, such as http://www.dominiondigital.com/.

```
<WebService(Namespace:="http://www.dominiondigital.com/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)>  _
Public Class CH14QuickWebServiceVB1
 Inherits System.Web.Services.WebService

 …

End Class 'CH14QuickWebServiceVB1
```

```
[WebService(Namespace = "http://www.dominiondigital.com/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class CH14QuickWebServiceCS1 : System.Web.Services.WebService
{
 …
} // CH14QuickWebServiceCS1
```

Visual Studio 2005 automatically adds a `WebServiceBinding` attribute to web service classes, as shown previously. This attribute indicates that the web service conforms to the Web Services Interoperability Basic Profile specification (WS-I BP 1.1). For more information on the web services interoperability, see http://www.ws-i.org .

To add useful functionality to the web service, you create methods as you would for any other class, except that you precede each method definition with a`WebMethod` attribute. The `WebMethod` attribute informs Visual Studio 2005 the method is to be exposed as part of the web service.

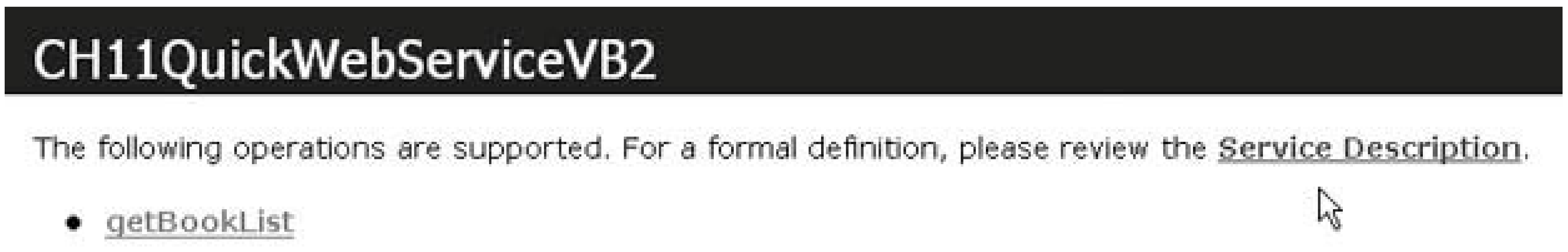
For instance, the `getBookList` method shown in our example queries a database for a list of books and returns the list in a `DataSet` . The number of books retrieved is defined by the`numberOfBooks` parameter. Nothing special is done in the code to support the web service. Visual Studio 2005 and th .NET Framework take care of the creation of the XML and the SOAP wrapper used to transfer the data to the client.

One of the big advantages to creating web services with Visual Studio 2005 and ASP.NET is the testing and debugging functionality provided. ASP.NET provides a series of web pages that create a test harness that can be used to test all the exposed methods of the web service. Visual Studio 2005 lets you set breakpoints in your web service code so you can step through it to verify its operation.

> The test harness created by Visual Studio 2005 can be used only if your web methods use .NET data types for the passed parameters.

To test a web service, run your project in Visual Studio 2005 in debug mode and access the *asmx* file for the web service. ASP.NET will display a page listing all of the methods exposed by the web service, as shown in Figure 14-1 .

Figure 14-1. Methods exposed by the web service

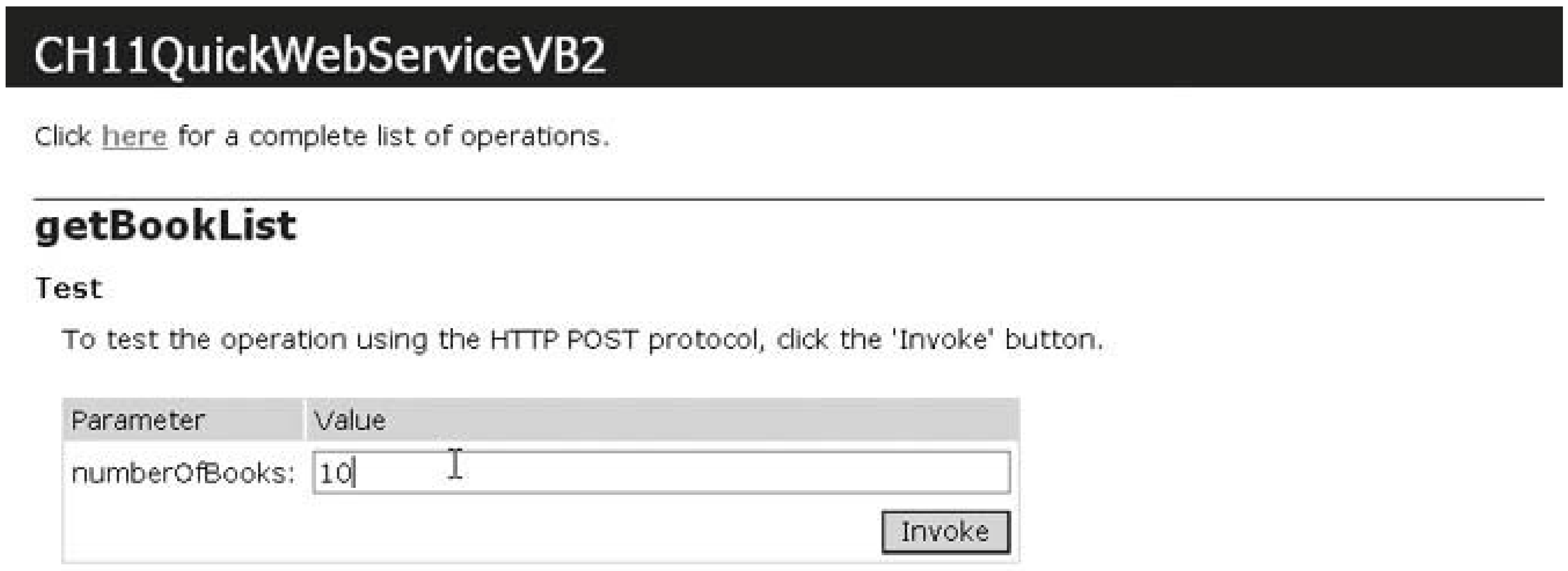

CH11QuickWebServiceVB2

The following operations are supported. For a formal definition, please review the Service Description.

- getBookList

In this example, clicking on the `getBookList` method displays another page where you can enter the required `numberOfBooks` parameter value and invoke (execute) the method. Though not shown in Figure 14-2 , this page displays the content of the XML-encoded request and response messages for the method using SOAP, Http Get, and Http Post. These samples allow you to examine the data that is exchanged when the method is called.

Figure 14-2. Invoking the method


CH11QuickWebServiceVB2

Click here for a complete list of operations.

getBookList

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
numberOfBooks:	10

When you click the Invoke button, ASP.NET generates an Http Get request and submits it to the web service. The web service responds with an Http response containing the requested data. In our example, the XML shown in Example 14-1is returned from the web service when the `numberOfBooks` parameter is set to `10` .

See Also

http://www.ws-i.org for information on Web Service Interoperability

Example 14-1. XML returned with numberOfBooks parameter set to 10

```

<?xml version="1.0" encoding="utf-8"?>
<DataSet xmlns="http://www.dominiondigital.com/">
 <xs:schema id="NewDataSet" xmlns=""
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="NewDataSet" msdata:IsDataSet="true"
       msdata:UseCurrentLocale="true">
   <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="Table">
    <xs:complexType>
    <xs:sequence>
      <xs:element name="Title" type="xs:string" minOccurs="0"/>
      <xs:element name="ISBN" type="xs:string" minOccurs="0"/>
      <xs:element name="Publisher" type="xs:string" minOccurs="0"/>
    </xs:sequence>
    </xs:complexType>
    </xs:element>
    </xs:choice>
   </xs:complexType>
  </xs:element>
 </xs:schema>
 <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">

  <NewDataSet xmlns="">
    <Table diffgr:id="Table1" msdata:rowOrder="0">
    <Title>.Net Framework Essentials</Title>
     <ISBN>0-596-00302-1</ISBN>
    <Publisher>O'Reilly</Publisher>
   </Table>
    <Table diffgr:id="Table2" msdata:rowOrder="1">
    <Title>Access Cookbook</Title>
     <ISBN>0-596-00084-7</ISBN>
    <Publisher>O'Reilly</Publisher>
   </Table>

    …

    <Table diffgr:id="Table10" msdata:rowOrder="9">
    <Title>Developing ASP Components</Title>
     <ISBN>1-565-92750-8</ISBN>
    <Publisher>O'Reilly</Publisher>
   </Table>
  </NewDataSet>
 </diffgr:diffgram>
</DataSet>
```

Example 14-2. Quick web service .asmx file (.vb)

```
<%@ WebService Language="VB"
 CodeBehind="~/App_Code/CH14QuickWebServiceVB1.vb"
 Class="VBWebServices.CH14QuickWebServiceVB1"  %>
```

Example 14-3. Quick web service code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

Namespace VBWebServices
 ''' <summary>
 ''' This class provides the code-behind for CH14QuickWebServiceVB1.asmx
 ''' </summary>
 <WebService(Namespace:="http://www.dominiondigital.com/")> _
 <WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)>  _
 Public Class CH14QuickWebServiceVB1
  Inherits System.Web.Services.WebService

  Public Sub CH11QuickWebServiceVB1()

  End Sub

  <WebMethod()> _
  Public Function HelloWorld() As String
   Return "Hello World"
  End Function

 End Class 'CH14QuickWebServiceVB1
End Namespace
```

Example 14-4. Quick web service .asmx file (.cs)

```
<%@ WebService Language="C#"
 CodeBehind="~/App_Code/CH14QuickWebServiceCS1.cs"
 Class="CSWebServices.CH14QuickWebServiceCS1"  %>
```

Example 14-5. Quick web service code-behind (.cs)

```
using System;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;

namespace CSWebServices
{
 /// <summary>
 /// This class provides the code-behind for CH14QuickWebServiceCS1.asmx
 /// </summary>
 [WebService(Namespace = "http://www.dominiondigital.com/")]
 [WebServiceBinding(ConformsTo  =  WsiProfiles.BasicProfile1_1)]
 public  class  CH14QuickWebServiceCS1  :  System.Web.Services.WebService
 {

   public  CH14QuickWebServiceCS1()
  {

  }

  [WebMethod]
  public string HelloWorld()
  {
   return "Hello World";
  }

 } //  CH14QuickWebServiceCS1
}
```

Example 14-6. Code-behind with method for obtaining a list of books (.vb

```
Option Explicit On
Option Strict On

Imports  System.Configuration
Imports  System.Data
Imports  System.Data.OleDb
Imports  System.Web
Imports  System.Web.Services
Imports  System.Web.Services.Protocols

Namespace  VBWebServices
 '''  <summary>
 '''  This  class  provides  the  code-behind  for  CH14QuickWebServiceVB2.asmx
 '''  </summary>
 <WebService(Namespace:="http://www.dominiondigital.com/")> _
 <WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)>  _
 Public  Class  CH14QuickWebServiceVB2
  Inherits System.Web.Services.WebService

  '''**********************************************************************
  ''' <summary>
  ''' This routine gets the list of books from the database.
  ''' </summary>
  '''
  ''' <param name="numberOfBooks">Set to the number of books to retrieve
  ''' </param>
  ''' <returns>DataSet containing the list of books</returns>
  <WebMethod()> _
  Function getBookList(ByVal numberOfBooks As Integer) As DataSet
   Dim dbConn As OleDbConnection = Nothing
   Dim da As OleDbDataAdapter = Nothing
   Dim dSet As DataSet = Nothing
   Dim cmdText As String
   Dim strSQL As String

   Try
    'get the connection string from web.config and open a connection
    'to the database
    cmdText = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDbConnection(cmdText)
    dbConn.Open()

    'build the query string used to get the data from the database
    strSQL = "SELECT Top " & numberOfBooks.ToString() & " " & _
     "Title, ISBN, Publisher " & _
     "FROM Book " & _
     "ORDER BY Title"

    'create a new dataset and fill it with the book data
    dSet = New DataSet
    da = New OleDbDataAdapter(strSQL, dbConn)
    da.Fill(dSet)

    'return the list of books
    Return (dSet)

   Finally
    'clean up
    If (Not IsNothing(dbConn)) Then
    dbConn.Close()
    End If
   End Try
  End Function 'getBookList
 End  Class  'CH14QuickWebServiceVB2
End Namespace
```

Example 14-7. Code-behind with method for obtaining a list of books (.cs

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web;
using System.Web.Services;

namespace CSWebServices
{
 /// <summary>
 /// This module provides the code behind for the CH14QuickWebServiceCS2.asmx
 /// </summary>
 [WebService(Namespace = "http://www.dominiondigital.com/")]
 [WebServiceBinding(ConformsTo  =  WsiProfiles.BasicProfile1_1)]
 public class CH14QuickWebServiceCS2 : System.Web.Services.WebService
 {
  ///**********************************************************************
  /// <summary>
  /// This routine gets the list of books from the database.
  /// </summary>
  ///
  /// <param name="numberOfBooks">Set to the number of books to retrieve
  /// </param>
  /// <returns>DataSet containing the list of books</returns>
  [WebMethod]
  public DataSet getBookList(int numberOfBooks)
  {
   OleDbConnection dbConn = null;
   OleDbDataAdapter da = null;
   DataSet dSet = null;
   String connectionStr = null;
   String cmdText = null;

   try
   {

    // get the connection string from web.config and open a connection
    // to the database
    connectionStr = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(connectionStr);
    dbConn.Open();

    //build the query string used to get the data from the database
    cmdText = "SELECT Top " + numberOfBooks.ToString() + " " +
      "Title, ISBN, Publisher " +
      "FROM Book " +
      "ORDER BY Title";

    // create a new dataset and fill it with the book data
    dSet = new DataSet();
    da = new OleDbDataAdapter(cmdText, dbConn);
    da.Fill(dSet);

    //return the list of books
    return (dSet);

   } // try

   finally
   {
    // cleanup
    if (dbConn != null)
    {
    dbConn.Close();
    }
   } // finally
  } // getBookList
 } //  CH14QuickWebServiceCS2
}
```

PREV

Recipe 14.3. Consuming a Web Service

Problem

You need to use a web service created by another group in your company to access data your application requires.

Solution

Add a web reference to an existing ASP.NET project using Visual Studio 2005. Create an instance of the web service class in your application and call its methods.

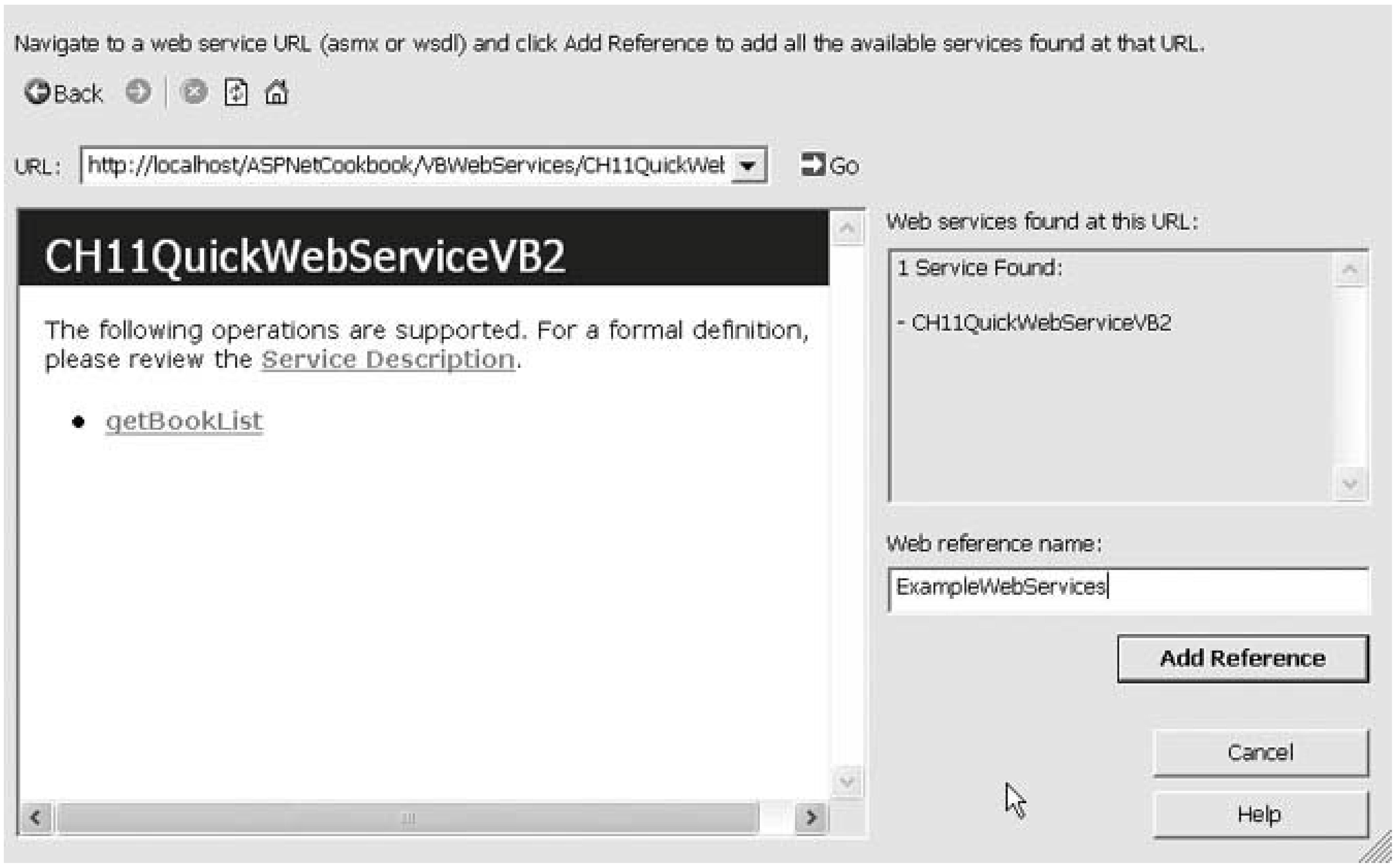
To add a web reference to an ASP.NET project in Visual Studio 2005:

1. Select the project in the Solution Explorer, right-click, and select Add Web Reference from the context menu.
2. In the Add Web Reference dialog box, enter the URL of the web service you want to consume, as shown in Figure 14-3, set the name for the web reference, and click the Add Reference button.

Visual Studio 2005 will create all the files needed to consume the web service and will place them in the *App_WebReferences* folder. The files include a *.disco* , a *.discomap* , and a *.wsdl* file for the web service.

After adding the web reference, create an instance of the web service class and call its methods in th code-behind class for the page. Examples 14-8, 14-9 through 14-10 show the *.aspx* file as well as VB and C# code-behind files for an example we've written to create an instance of the web service class from Recipe 14.1 and call its methods.

Figure 14-3. Adding a web reference


Navigate to a web service URL (asmx or wsdl) and click Add Reference to add all the available services found at that URL.
Back
URL: http://localhost/ASPNetCookbook/VBWebServices/CH11QuickWel
Go
CH11QuickWebServiceVB2
The following operations are supported. For a formal definition, please review the Service Description.
getBookList
Web services found at this URL:
1 Service Found:
- CH11QuickWebServiceVB2
Web reference name:
ExampleWebServices
Add Reference
Cancel
Help

Discussion

Visual Studio 2005 makes consumption of a web service easy by creating all of the plumbing for you. You don't have to worry about creating proxy classes and the SOAP messages; it's all done for you when you add a web reference.

In our example that illustrates the solution, the web service created in Recipe 14.1 is used to obtain a list of books and display the list in a `DataGrid`. We have accomplished this by writing just a few lines of code.

The first step in consuming a web service is to add a web reference to your project by selecting the project in the Solution Explorer, right-clicking, and selecting Add Web Reference from the context menu. You need to enter the URL of the web service you want to consume. This is normally the full URL of the *.asmx* file or the WSDL file of the web service. When the web service resides on your own computer, as is the case with our example for this recipe, a URL such as the following will do:

http://localhost/ASPNetCookbook/VBWebServices/CH14QuickWebServiceVB2.asmx

When adding a web reference, select Add Web Reference *not* Add Reference. Add Reference is used to add a reference to an assembly on the local server. Add Web Reference is used to add a reference to a web service.

After entering the URL of the *.asmx* or WSDL file of the web service, the Add Web Reference dialog box displays the operations provided by the web service in the left pane.

The web reference name should be changed to remove the tight coupling to a specific server becaus the server hosting the web service can change. For our example, we have renamed the web reference to `ExampleWebServices` , as shown in Figure 14-3.

When you click Add Reference, Visual Studio 2005 creates and adds to your project all the files needed to make a web service available to your application, including a disco file, which helps the application locate the service, and a WSDL file, which defines the services available from the web service. Then, it creates a proxy class that you use to access the web service. (The*proxy* class interfaces with the web service and provides a local representation of the service.) The web referenc is given the name provided in the Add Web Reference dialog box.

> Visual Studio 2003 created a physical class called`Reference` as the proxy class. Visual Studio 2005 does not create the physical class but generates a "virtual" proxy class that is not visible in the project.

After you add a web reference to the service you plan to employ, you need to create an instance of the proxy class. For example, here's how we create an instance of the proxy class for the example web service in Recipe 14.1:

```
bookServices = New ExampleWebServices.CH14QuickWebServiceVB2
```

```
bookServices = new ExampleWebServices.CH14QuickWebServiceCS2();
```

In our example, the `getBookList` method is called to obtain a `DataSet` containing the book list. Though this method call looks like a standard method call, the proxy class is calling the web service to get the data.

```
books = bookServices.getBookList(NUMBER_OF_BOOKS)
```

```
books = bookServices.getBookList(NUMBER_OF_BOOKS);
```

For our example, the `DataSet` returned by the web service is bound to a `DataGrid` on the form to display the list of books:

```
dgBooks.DataSource = books
dgBooks.DataBind()
```

```
 dgBooks.DataSource = books;
 dgBooks.DataBind();
```

By creating all the plumbing required to access web services, which would be tedious to write and error-prone if you had to do it yourself, Visual Studio 2005 makes using web services in your applications more practical. This is true if the web service provider and consumer are both .NET implementations, because most of the data types provided by the Common Language Runtime (CLR) can be used in the web service interfaces. (See the ".NET Web Service Idiosyncrasies" sidebar in Recipe 14.3 for more on this topic.)

Web services provided by other technology platforms, such as Java, can be consumed by .NET applications. Java and other platforms do not have a set of rich data types that match the CLR data types, so the interface must be designed using simple data types. The details ofconsuming a web service from other technologies are beyond the scope of this book.

See Also

Recipe 14.1; if you need to use web services created with Java, see*Java Web Services* , by Dave Chappell and Tyler Jewell (O'Reilly).

Example 14-8. Consuming a web service (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH14ConsumingAWebServiceVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH14ConsumingAWebServiceVB"
 Title="Consuming A WebService" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Consuming a Web Service (VB)
 </div>
 <div align="center">
  <asp:DataGrid id="dgBooks"
     runat="server"
      BorderColor="000080"

     BorderWidth="2px"
    AutoGenerateColumns="True"
    width="90%" />
 </div>
</asp:Content>
```

Example 14-9. Consuming a web service code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Data

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code behind for
 ''' CH14ConsumingAWebServiceVB.aspx
 ''' </summary>
 Partial Class CH14ConsumingAWebServiceVB
  Inherits System.Web.UI.Page
  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Const NUMBER_OF_BOOKS As Integer = 10

   Dim bookServices As ExampleWebServices.CH14QuickWebServiceVB2
   Dim books As DataSet

   If (Not Page.IsPostBack) Then
    'create an instance of the web service proxy class
    bookServices = New ExampleWebServices.CH14QuickWebServiceVB2

    'get the books from the service
    books = bookServices.getBookList(NUMBER_OF_BOOKS)

    'bind the book list to the datagrind on the form
    dgBooks.DataSource = books
    dgBooks.DataBind()
   End If
  End Sub 'Page_Load
 End Class 'CH14ConsumingAWebServiceVB
End Namespace
```

Example 14-10. Consuming a web service code-behind (.cs)

```
using System;
using System.Data;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH14ConsumingAWebServiceCS.aspx
 /// </summary>
 public partial class CH14ConsumingAWebServiceCS : System.Web.UI.Page
 {
   ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   const int NUMBER_OF_BOOKS = 10;

   ExampleWebServices.CH14QuickWebServiceCS2 bookServices = null;
   DataSet books = null;

   if (!Page.IsPostBack)
   {
    // create an instance of the web service proxy class
    bookServices = new ExampleWebServices.CH14QuickWebServiceCS2();

    // get the books from the service
    books = bookServices.getBookList(NUMBER_OF_BOOKS);

    // bind the book list to the datagrind on the form
    dgBooks.DataSource = books;
    dgBooks.DataBind();
   }
  } // Page_Load
 } // CH14ConsumingAWebServiceCS
}
```

Recipe 14.4. Creating a Web Service That Returns a Custom Object

Problem

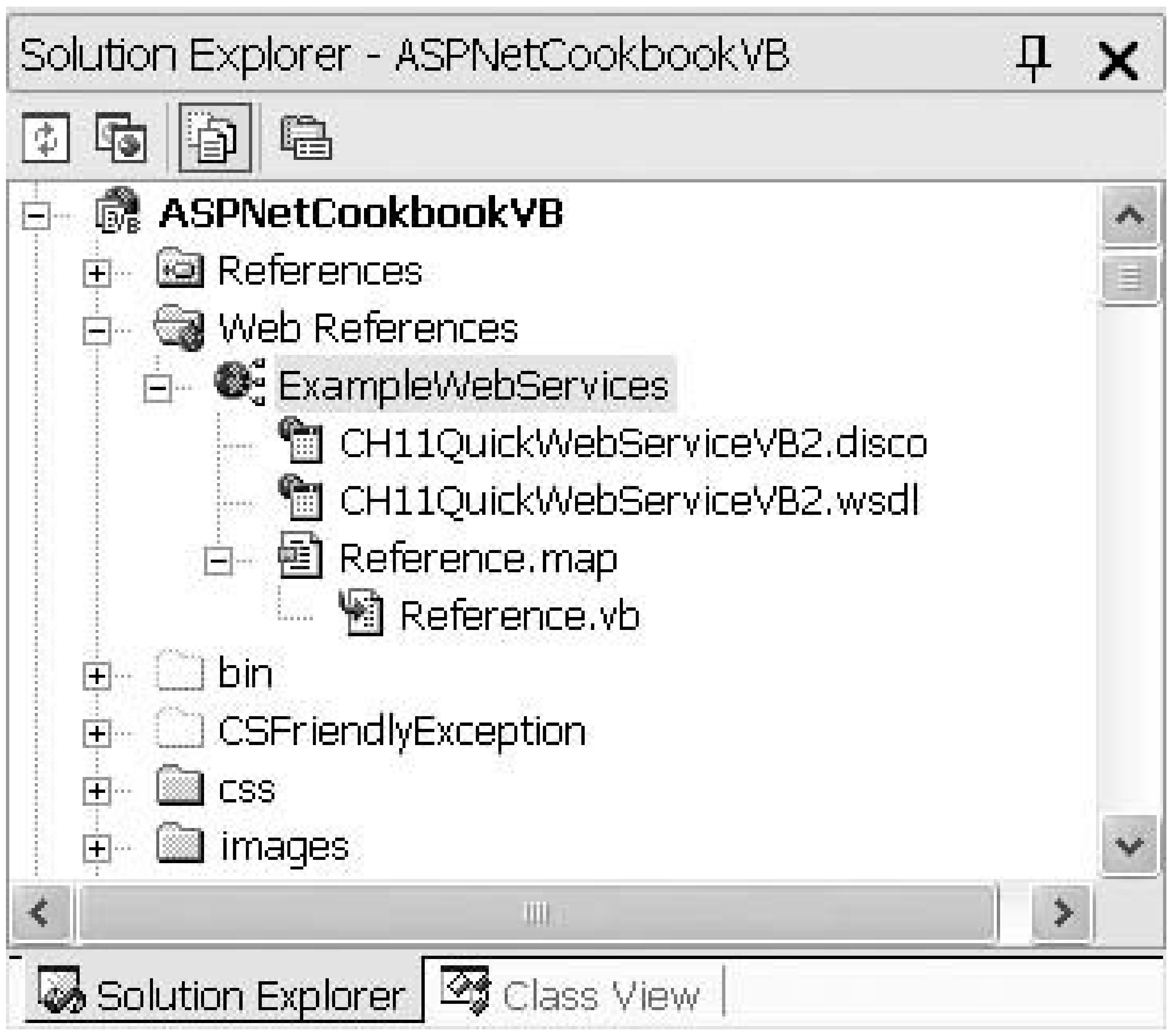
You want to create a web service that returns a custom object because none of the .NET data types meets your needs.

Solution

Create a class that encapsulates the data you need and use it as the return type of a method of your web service.

To demonstrate this solution, we have created the custom class, `BookData`, shown in Examples 14-11 (VB) and 14-12 (C#). The class encapsulates information about books stored in a database. A class that uses a web service that returns book information from a database using the custom class is shown in Examples 14-13 (VB) and 14-14 (C#). Examples 14-15, 14-16 through 14-17 show the *.aspx* file and VB and C# code-behind files for our application that demonstrates how we use the web service. Figure 14-4 shows the Solution Explorer in Visual Studio 2005 with the files used for this recipe and other recipes in this chapter.

Figure 14-4. Solution Explorer showing files for this recipe



Discussion

Web services use XML to transfer data, and rely on the CLR to serialize most data types to XML. If you create an object that contains public properties or variables of the types the CLR can serialize, the CLR will serialize the object for you with no additional coding when it is used as the return type o a web service.

A custom object to be returned by a web service must meet two requirements. First, the object must contain only data types that can be serialized (they must implement the`ISerializable` interface). All of the .NET base data types and the majority of its complex data types can be serialized. The notable exceptions are `DataTable` (in ASP.NET 1.x), `DataRow`, and `DataView`.

Second, the class defining the object must include a public default constructor (one with no parameters). The CLR uses this constructor to serialize the object.

Let's take a look at the code we have written to illustrate this solution. To begin with, we have created the class shown in Examples 14-11 (VB) and 14-12 (C#), which encapsulates the data that a method of the web service will return. The class consists of the four sections shown inTable 14-1.

Table 14-1. Elements of the class returned by the sample web service

Element	Description
Private attributes	Used to store the object data

Element	Description
Public properties	Used to access the object data
First constructor	Used to create the object and populate it with data for a specific book
Second constructor	Provides the default public constructor required for serialization and provides the ability to create an object with all default values

The class shown in Examples 14-13 (VB) and 14-14 (C#) implements our web service. To make our example more useful, the `getBookList` method described in Recipe 14.1 is included in this class. For more on the `getBookList` web service method, refer to Recipe 14.1.

In our example, the `getBookData` method creates an instance of the `BookData` class and uses it as the return type for the method. The ID of the required book is passed to provide the constructor the information it needs to create the object and populate it with the requested book data:

VB

```
bookInfo = New BookData(bookID)
Return (bookInfo)
```

```
bookInfo = new BookData(bookID);
return (bookInfo);
```

Our *.aspx* file, shown in Example 14-15, uses a `ListBox` to display a list of available books and a group of `Literal` controls to display the title, ISBN, and other details about the book.

The `Page_Load` method in our example's code-behind, shown in Examples 14-16 (VB) and 14-17 (C#), is responsible for populating the `ListBox` with the list of available books. Our first step is to create an instance of the web service proxy class:

```
bookServices = New ExampleBookServices.CH14BookServicesVB
```

```
bookServices = new ExampleBookServices.CH14BookServicesCS();
```

In our example, the web reference was renamed `ExampleBookServices` and the class that implements the web service is named `CH14BookServicesVB` (or `CH14BookServicesCS`).

Our next step is to call the getBookList method of the proxy class to get a list of available books:

VB
```
books = bookServices.getBookList(NUMBER_OF_BOOKS)
```

C#
```
books = bookServices.getBookList(NUMBER_OF_BOOKS);
```

After getting the list, the data is bound to the ListBox by setting the DataSource property to the DataSet containing the list of books. The DataTextField is set to the column in the DataSet containing the title of the book to define what will be displayed in the ListBox. The DataValueField is set to the column in the DataSet containing the BookID to identify the book when an item is selected. Finally, the DataBind method is called to bind the data in the DataSet to the ListBox:

VB
```
lstBooks.DataSource = books
lstBooks.DataTextField = "Title"
lstBooks.DataValueField = "BookID"
lstBooks.DataBind()
```

```
lstBooks.DataSource = books;
lstBooks.DataTextField = "Title";
lstBooks.DataValueField = "BookID";
lstBooks.DataBind();
```

Rather than attempting to handle the case where the user does not select a book in the ListBox, we have simplified our example by selecting the first book in the list and then calling the getBookDetails method (described next):

```
lstBooks.SelectedIndex = 0
getBookDetails()
```

```
lstBooks.SelectedIndex = 0;
getBookDetails();
```

The getBookDetails method is responsible for calling the web service that retrieves the details of the selected book. The getBookData method that is exposed by the web service is responsible for returning a custom object.

First, the method gets the ID of the selected book from the ListBox:

VB
```
bookID = CInt(lstBooks.SelectedItem.Value)
```

C#
```
bookID  =  System.Convert.ToInt32(lstBooks.SelectedItem.Value);
```

Next, an instance of the web service proxy class is created and the `getBookData` method is called to get the details of the book from the web service:

VB
```
bookServices  =  New  ExampleBookServices.CH14BookServicesVB
bookInfo = bookServices.getBookData(bookID)
```

C#
```
bookServices  =  new  ExampleBookServices.CH14BookServicesCS();
bookInfo = bookServices.getBookData(bookID);
```

The controls used to display the book details are initialized with the data in the `BookData` object returned from the web service:

```
litTitle.Text = bookInfo.title
litIsbn.Text = bookInfo.Isbn
litDescription.Text = bookInfo.description
litPublisher.Text = bookInfo.publisher
litListPrice.Text  =  bookInfo.listPrice.ToString("0.00")
litDate.Text = bookInfo.publishDate.ToShortDateString()
```

```
litTitle.Text = bookInfo.title;
litIsbn.Text = bookInfo.Isbn;
litDescription.Text = bookInfo.description;
litPublisher.Text = bookInfo.publisher;
litListPrice.Text  =  bookInfo.listPrice.ToString("0.00");
litDate.Text = bookInfo.publishDate.ToShortDateString();
```

One of the primary benefits of encapsulating data as we have in this example is the improvement in the performance of the web service. By returning the custom `BookData` object from our web service, all of the data is retrieved in a single call. If each piece of data for the book is obtained using separate calls, the application's performance will decrease. When you use a web service to retrieve data, return the largest possible block of data for each call. This reduces the significant overhead of serializing the data to XML and wrapping it with SOAP and HTTP protocol wrappers.

See Also

Recipe 14.1

.NET Web Service Idiosyncrasies

Web services produced and consumed by .NET applications can use any of the data types of the CLR that implement the `ISerializable` interface. If your web services need to be consumed by other technologies, such as Java, it will be necessary to limit the data types to simple types, such as integer, string, etc. In addition, .NET web services use Document Literal encoding by default, while many other technologies use RPC encoding. These factors must be considered when developing web services that are produced and consumed by different technologies.

Web services return data only

Web services let you pass data between applications; however, they cannot pass behavior. If you create a typical class that provides methods to act on the data in the instantiated object and use it as the return type from a web service method, you will find the methods to act on the data are unavailable when the object is returned. Though the object created by the web service and the object created by the application consuming the web service have the same name, they are not created from the same class. The `BookData` proxy class (VB version) created for our example is shown here and does not resemble the `BookData` class shown in Example 14-11. Essentially, it is a structure containing the data exposed by the class returned by the web service.

```
<System.Xml.Serialization.XmlTypeAttribute([Namespace]:="http://
  www.dominiondigital.com")> _
 Public Class BookData

  '<remarks/>
  Public title As String

  '<remarks/>
  Public Isbn As String

  '<remarks/>
  Public description As String

  '<remarks/>
  Public publisher As String

  '<remarks/>
  Public listPrice As Single

  '<remarks/>
```

```
   Public publishDate As Date

   '<remarks/>
   Public bookID As Integer
 End Class
```

Example 14-11. Custom class (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace VBWebServices
 ''' <summary>
 ''' This class provides the data class used to encapsulate book data
 ''' returned from a web service
 ''' </summary>
 Public Class BookData
  'private attributes
  Private mBookID As Integer
  Private mTitle As String
  Private mIsbn As String
  Private mDescription As String
  Private mPublisher As String
  Private mListPrice As Single
  Private mPublishDate As DateTime

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the book ID
  ''' </summary>
  Public Property bookID() As Integer
   Get
    Return (mBookID)
   End Get
   Set(ByVal Value As Integer)
    mBookID = Value
   End Set
  End Property 'bookID

  '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the title
  ''' </summary>
  Public Property title() As String
   Get
    Return (mTitle)
   End Get
   Set(ByVal Value As String)
    mTitle = Value
   End Set
  End Property 'title

   '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the ISBN

  ''' </summary>
  Public Property Isbn() As String
   Get
    Return (mIsbn)
   End Get
   Set(ByVal Value As String)
    mIsbn = Value
   End Set
  End Property 'Isbn

   '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the description
  ''' </summary>
  Public Property description() As String
   Get
    Return (mDescription)
   End Get
   Set(ByVal Value As String)
    mDescription = Value
   End Set
  End Property 'description

   '''*************************************************************************
  ''' <summary>
  ''' This property provides the ability to get/set the publisher
  ''' </summary>
  Public Property publisher() As String
   Get
    Return (mPublisher)
   End Get
   Set(ByVal Value As String)
    mPublisher = Value
   End Set
  End Property 'publisher

   '''*************************************************************************
```

```
''' <summary>
''' This property provides the ability to get/set the listPrice
''' </summary>
Public Property listPrice() As Single
 Get
  Return (mListPrice)
 End Get
 Set(ByVal Value As Single)
  mListPrice = Value
 End Set
End Property 'listPrice

'''*************************************************************************
''' <summary>
''' This property provides the ability to get/set the publishDate

''' </summary>
Public Property publishDate() As DateTime
 Get
  Return (mPublishDate)
 End Get
 Set(ByVal Value As Date)
  mPublishDate = Value
 End Set
End Property 'publishDate

'''*************************************************************************
''' <summary>
''' This constructor creates the object and populates it with data for
''' the passed book ID.
''' </summary>
'''
''' <param name="ID">Set to the ID of the book from which to create the
''' object
''' </param>
Public Sub New(ByVal ID As Integer)
 Dim dbConn As OleDbConnection = Nothing
 Dim da As OleDbDataAdapter = Nothing
 Dim dTable As DataTable = Nothing
 Dim dRow As DataRow = Nothing
 Dim connectionStr As String
 Dim cmdText As String

 Try
  'get the connection string from web.config and open a connection
  'to the database
  connectionStr = ConfigurationManager. _
  ConnectionStrings("dbConnectionString").ConnectionString
  dbConn = New OleDbConnection(connectionStr)
  dbConn.Open()

  'build the query string used to get the data from the database
```

```
        cmdText = "SELECT BookID, Title, ISBN, Description, Publisher, " & _
          "ListPrice, PublishDate " & _
          "FROM Book " & _
          "WHERE BookID=" & ID.ToString()

        'create a new data table and fill it with the book data
        dTable = New DataTable
        da = New OleDbDataAdapter(cmdText, dbConn)
        da.Fill(dTable)

        'populate object with the book data read from the database
        dRow = dTable.Rows(0)
        bookID = CInt(dRow.Item("BookID"))
        title = CStr(dRow.Item("Title"))
        Isbn = CStr(dRow.Item("ISBN"))
        description = CStr(dRow.Item("Description"))

        publisher = CStr(dRow.Item("Publisher"))
        listPrice = CSng(dRow.Item("ListPrice"))
        publishDate = CDate(dRow.Item("PublishDate"))

      Finally
        'clean up
        If (Not IsNothing(dbConn)) Then
        dbConn.Close()
        End If
      End Try
    End Sub 'New

    '*************************************************************************
    '
    ' ROUTINE: New
    '
    ' DESCRIPTION: This constructor creates the object will default values
    '-------------------------------------------------------------------------
    Public Sub New()
      bookID = -1
      title = ""
      Isbn = ""
      description = ""
      publisher = ""
      listPrice = 0
      publishDate = DateTime.MinValue
    End Sub 'New
  End Class 'BookData
End Namespace
```

Example 14-12. Custom class (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web;

namespace CSWebServices
{
 /// <summary>
 /// This class provides the data class used to encapsulate book data
 /// returned from a web service
 /// </summary>
 public class BookData
 {
  // private attributes
  private int mBookID;
  private string mTitle;
  private string mIsbn;
  private string mDescription;
  private string mPublisher;

  private Decimal mListPrice;
  private DateTime mPublishDate;

  ///**********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the book ID
  /// </summary>
  public int bookID
  {
   get
   {
    return (mBookID);
   }
   set
   {
    mBookID = value;
   }
  } // bookID

  ///**********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the title
  /// </summary>
  public string title
  {
   get
   {
    return (mTitle);
   }
   set
   {
    mTitle = value;
   }
  } // title

  ///*********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the ISBN
  /// </summary>
  public string Isbn
  {
   get
   {
    return (mIsbn);
   }
   set
   {
    mIsbn = value;
   }
  } // Isbn

  ///*********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the description
  /// </summary>
  public string description
  {
   get
   {
    return (mDescription);
   }
   set
   {
    mDescription = value;
   }
  } // description

  ///*********************************************************************
  /// <summary>
  /// This property provides the ability to get/set the publisher
  /// </summary>
  public string publisher
  {
   get
   {
    return (mPublisher);
   }
   set
   {
    mPublisher = value;
   }
  } // publisher
```

```
 ///*********************************************************************
/// <summary>
/// This property provides the ability to get/set the listPrice
/// </summary>
public Decimal listPrice
{
 get
 {
  return (mListPrice);
 }
 set
 {
  mListPrice = value;
 }
} // listPrice

 ///*********************************************************************
/// <summary>
/// This property provides the ability to get/set the publishDate
/// </summary>

public DateTime publishDate
{
 get
 {
  return (mPublishDate);
 }
 set
 {
  mPublishDate = value;
 }
} // publishDate

 ///*********************************************************************
/// <summary>
/// This constructor creates the object and populates it with data for
/// the passed book ID.
/// </summary>
///
/// <param name="ID">Set to the ID of the book from which to create the
/// object
/// </param>
public BookData(int ID)
{
 OleDbConnection dbConn = null;
 OleDbDataAdapter da = null;
 DataTable dTable = null;
 DataRow dRow = null;
 string connectionStr = null;
 string cmdText = null;
```

```
  try
  {
   // get the connection string from web.config and open a connection
   // to the database
   connectionStr = ConfigurationManager.
   ConnectionStrings["dbConnectionString"].ConnectionString;
   dbConn = new OleDbConnection(connectionStr);
   dbConn.Open();

   //build the query string used to get the data from the database
   cmdText = "SELECT BookID, Title, ISBN, Description, Publisher, " +
     "ListPrice, PublishDate " +
     "FROM Book " +
     "WHERE BookID=" + ID.ToString();

   // create a new data table and fill it with the book data
   dTable = new DataTable();
   da = new OleDbDataAdapter(cmdText, dbConn);
   da.Fill(dTable);

   // populate object with the book data read from the database
    dRow = dTable.Rows[0];

   bookID = (int)(dRow["BookID"]);
   title = (string)(dRow["Title"]);
   Isbn = (string)(dRow["ISBN"]);
   description = (string)(dRow["Description"]);
   publisher = (string)(dRow["Publisher"]);
   listPrice = (Decimal)(dRow["ListPrice"]);
   publishDate = (DateTime)(dRow["PublishDate"]);
  }

  finally
  {
   // cleanup
   if (dbConn != null)
   {
   dbConn.Close();
   }
  } // finally
 }

 //*************************************************************************
 //
 // ROUTINE: BookData
 //
 // DESCRIPTION: This constructor creates the object will default values
 //------------------------------------------------------------------------
 public BookData()
 {
  bookID = -1;
  title = "";
```

```
    Isbn = "";
    description = "";
    publisher = "";
    listPrice = 0;
    publishDate = DateTime.MinValue;
   }
  } // BookData
}
```

Example 14-13. Web service returning a custom object (.vb)

```
Option Explicit On
Option Strict On

Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

Namespace VBWebServices
 ''' <summary>

 ''' This class provides the code-behind for CH14BookServicesVB.asmx
 ''' </summary>
 <WebService(Namespace:="http://www.dominiondigital.com/")> _
 <WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)>   _
 Public Class CH14BookServicesVB
  Inherits System.Web.Services.WebService

   '''**********************************************************************
  ''' <summary>
  ''' This routine gets the list of books from the database.
  ''' </summary>
  '''
  ''' <param name="numberOfBooks">Set to the number of books to retrieve
  ''' </param>
  ''' <returns>DataSet containing the list of books</returns>
  <WebMethod()> _
  Function getBookList(ByVal numberOfBooks As Integer) As DataSet
   Dim dbConn As OleDbConnection = Nothing
   Dim da As OleDbDataAdapter = Nothing
   Dim dSet As DataSet = Nothing
   Dim connectionStr As String
   Dim cmdText As String
```

```
    Try
     'get the connection string from web.config and open a connection
     'to the database
     connectionStr = ConfigurationManager. _
     ConnectionStrings("dbConnectionString").ConnectionString
     dbConn = New OleDbConnection(connectionStr)
     dbConn.Open()

     'build the query string used to get the data from the database
     cmdText = "SELECT Top " & numberOfBooks.ToString() & " " & _
       "BookID, Title, ISBN, Publisher " & _
       "FROM Book " & _
       "ORDER BY Title"

     'create a new dataset and fill it with the book data
     dSet = New DataSet
     da = New OleDbDataAdapter(cmdText, dbConn)
     da.Fill(dSet)

     'return the list of books
     Return (dSet)

    Finally
     'clean up
     If (Not IsNothing(dbConn)) Then
     dbConn.Close()
     End If
    End Try
   End Function 'getBookList

   '''**************************************************************************
   ''' <summary>
   ''' This routine gets the data for the passed book
   ''' </summary>
   '''
   ''' <param name="bookID">Set to the ID of the book for which the data is
   ''' required
   ''' </param>
   ''' <returns>BookData containing the data for the requested book</returns>
   <WebMethod()> _
   Function getBookData(ByVal bookID As Integer) As BookData

    Dim bookInfo As BookData

    'create a new BookData object containing the requested data
    bookInfo = New BookData(bookID)
    Return (bookInfo)
   End Function ' getBookData
  End Class 'CH14BookServicesVB
End Namespace
```

Example 14-14. Web service returning a custom object (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web;
using System.Web.Services;

namespace CSWebServices
{
  /// <summary>
  /// This module provides the code behind for the CH14BookServicesCS.asmx
  /// </summary>
  [WebService(Namespace = "http://www.dominiondigital.com/")]
  [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
  public class CH14BookServicesCS : System.Web.Services.WebService
  {
  ///***********************************************************************
  /// <summary>
  /// This routine gets the list of books from the database.
  /// </summary>
  ///
  /// <param name="numberOfBooks">Set to the number of books to retrieve
  /// </param>
  /// <returns>DataSet containing the list of books</returns>
  [WebMethod]
  public DataSet getBookList(int numberOfBooks)
  {
   OleDbConnection dbConn = null;
   OleDbDataAdapter da = null;

   DataSet dSet = null;
   String connectionStr = null;
   String cmdText = null;

   try
   {
    // get the connection string from web.config and open a connection
    // to the database
    connectionStr = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(connectionStr);
    dbConn.Open();

    //build the query string used to get the data from the database
    cmdText = "SELECT Top " + numberOfBooks.ToString() + " " +
      "BookID, Title, ISBN, Publisher " +
      "FROM Book " +
      "ORDER BY Title";

    // create a new dataset and fill it with the book data
    dSet = new DataSet();
    da = new OleDbDataAdapter(cmdText, dbConn);
    da.Fill(dSet);

    //return the list of books
    return (dSet);
   } // try

   finally
   {
    // cleanup
    if (dbConn != null)
    {
    dbConn.Close();
    }
   } // finally
  } // getBookList

  ///*************************************************************************
  /// <summary>
  /// This routine gets the data for the passed book
  /// </summary>
  ///
  /// <param name="bookID">Set to the ID of the book for which the data is
  /// required
  /// </param>
  /// <returns>BookData containing the data for the requested book</returns>
  [WebMethod]
  public BookData getBookData(int bookID)
  {
   BookData bookInfo = null;

   // create a new BookData object containing the requested data
   bookInfo = new BookData(bookID);
   return (bookInfo);
  } // getBookData
 } //  CH14BookServicesCS
}
```

Example 14-15. Using the web service returning a custom object (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH14CustomObjectWithWebServiceVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH14CustomObjectWithWebServiceVB"
 Title="Web Service Returning A Custom Object" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Returning a Custom Object From a Web Service (VB)
 </div>
 <table width="90%" align="center" border="0">
  <tr>
   <td align="center" class="MenuItem">
    <br />Available Books<br /><br />
    <asp:ListBox ID="lstBooks" Runat="server"
     AutoPostBack="True"
     OnSelectedIndexChanged="lstBooks_SelectedIndexChanged" />
   </td>
  </tr>
  <tr>
   <td>
    <br />
     <table width="60%" align="center"
       border="1"
        cellpadding="5" cellspacing="0" class="MenuItem">
    <tr>
     <td width="25%" align="right">Title: </td>
     <td width="75%">
    <asp:Literal ID="litTitle" Runat="server" /></td>
    </tr>
    <tr>
     <td width="25%" align="right">ISBN: </td>
     <td width="75%">
    <asp:Literal ID="litIsbn" Runat="server" /></td>
    </tr>
    <tr>
     <td width="25%" align="right">Description: </td>
     <td width="75%">
    <asp:Literal ID="litDescription" Runat="server" /></td>
    </tr>
    <tr>
     <td width="25%" align="right">Publisher: </td>
     <td width="75%">
    <asp:Literal ID="litPublisher" Runat="server" /></td>

    </tr>
    <tr>
     <td width="25%" align="right">List Price: </td>
     <td width="75%">
    <asp:Literal ID="litListPrice" Runat="server" /></td>
    </tr>
    <tr>
     <td width="25%" align="right">Date: </td>
     <td width="75%">
    <asp:Literal ID="litDate" Runat="server" /></td>
    </tr>
    </table>
   </td>
  </tr>
   </table>
</asp:Content>
```

Example 14-16. Using the web service returning a custom object code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Data

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code behind for
  '''  CH14CustomObjectWithWebServiceVB.aspx
  ''' </summary>
  Partial Class CH14CustomObjectWithWebServiceVB
    Inherits System.Web.UI.Page
    '''*************************************************************************
    ''' <summary>
    ''' This routine provides the event handler for the page load event. It
    ''' is responsible for initializing the controls on the page.
    ''' </summary>
    '''
    ''' <param name="sender">Set to the sender of the event</param>
    ''' <param name="e">Set to the event arguments</param>
    Private Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
      Const NUMBER_OF_BOOKS As Integer = 20

      Dim bookServices As ExampleBookServices.CH14BookServicesVB
      Dim books As DataSet

      If (Not Page.IsPostBack) Then
        'create an instance of the web service proxy class
        bookServices = New ExampleBookServices.CH14BookServicesVB

        'get the books from the service
        books = bookServices.getBookList(NUMBER_OF_BOOKS)

        'bind the book list to the listbox on the form
        lstBooks.DataSource = books
        lstBooks.DataTextField = "Title"
```

```
     lstBooks.DataValueField = "BookID"
     lstBooks.DataBind()

     'select the first item in the list and get the details
      lstBooks.SelectedIndex = 0
     getBookDetails()
   End If
 End Sub 'Page_Load

 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the book listbox selected
 ''' index changed event. It is responsible for getting the book data
 ''' for the selected book
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub lstBooks_SelectedIndexChanged(ByVal sender As Object, _
         ByVal e As System.EventArgs)
   'get the data for the selected book
   getBookDetails()
 End Sub 'lstBooks_SelectedIndexChanged

 '''*************************************************************************
 ''' <summary>
 ''' This routine gets the details for the currently selected book and
 ''' initializes the controls on the with the data.
 ''' </summary>
 Private Sub getBookDetails()
   Dim bookServices As ExampleBookServices.CH14BookServicesVB
   Dim bookInfo As ExampleBookServices.BookData
   Dim bookID As Integer

   'get the currently selected book
   bookID = CInt(lstBooks.SelectedItem.Value)

   'create an instance of the web service proxy class
   'and get the book data
   bookServices = New ExampleBookServices.CH14BookServicesVB
   bookInfo = bookServices.getBookData(bookID)

   'set the controls on the form to display the book data
   litTitle.Text = bookInfo.title
   litIsbn.Text = bookInfo.Isbn
   litDescription.Text = bookInfo.description
   litPublisher.Text = bookInfo.publisher
   litListPrice.Text = bookInfo.listPrice.ToString("0.00")
   litDate.Text = bookInfo.publishDate.ToShortDateString()

   End Sub 'getBookDetails
End Class 'CH14CustomObjectWithWebServiceVB
End Namespace
```

Example 14-17. Using the web service returning a custom object code-behind (.cs)

```
using System;
using System.Data;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code behind for
 ///  CH14CustomObjectWithWebServiceCS.aspx
 /// </summary>
 public partial class CH14CustomObjectWithWebServiceCS : System.Web.UI.Page
 {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {

   const int NUMBER_OF_BOOKS = 20;

   ExampleBookServices.CH14BookServicesCS bookServices = null;
   DataSet books = null;

   if (!Page.IsPostBack)
   {
    // create an instance of the web service proxy class
     bookServices = new ExampleBookServices.CH14BookServicesCS();

    // get the books from the service
    books = bookServices.getBookList(NUMBER_OF_BOOKS);

    // bind the book list to the listbox on the form
    lstBooks.DataSource = books;
    lstBooks.DataTextField = "Title";
    lstBooks.DataValueField = "BookID";
    lstBooks.DataBind();

    // select the first item in the list and get the details
```

```
     lstBooks.SelectedIndex = 0;
    getBookDetails();
   }
  } // Page_Load

  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the book listbox selected
  /// index changed event. It is responsible for getting the book data
  /// for the selected book
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void lstBooks_SelectedIndexChanged(Object sender,
           System.EventArgs e)
  {
   // get the data for the selected book
   getBookDetails();
  } // lstBooks_SelectedIndexChanged

  ///***********************************************************************
  /// <summary>
  /// This routine gets the details for the currently selected book and
  /// initializes the controls on the with the data.
  /// </summary>
  private void getBookDetails()
  {
   ExampleBookServices.CH14BookServicesCS bookServices = null;
   ExampleBookServices.BookData bookInfo = null;
   int bookID;

   // get the currently selected book
   bookID = System.Convert.ToInt32(lstBooks.SelectedItem.Value);

   // create an instance of the web service proxy class
   // and get the book data
   bookServices = new ExampleBookServices.CH14BookServicesCS();
   bookInfo = bookServices.getBookData(bookID);

   // set the controls on the form to display the book data
   litTitle.Text = bookInfo.title;
   litIsbn.Text = bookInfo.Isbn;
   litDescription.Text = bookInfo.description;
   litPublisher.Text = bookInfo.publisher;
   litListPrice.Text = bookInfo.listPrice.ToString("0.00");
   litDate.Text = bookInfo.publishDate.ToShortDateString();
  } // getBookDetails
 } // CH14CustomObjectWithWebServiceCS
}
```

PREV

PREV

Recipe 14.5. Setting the URL of a Web Service at Runtime

Problem

You need to set the URL of the web service at runtime.

Solution

In the code-behind class for your page, set the URL property of the proxy class to the required URL after instantiating the proxy object, as shown here:

VB

```
bookServices  =  New  ExampleBookServices.CH14BookServicesVB
bookServices.Url = _
    "http://michaelk2/aspnetcookbook2vb/VBWebServices/CH14BookServicesVB.asmx"
```

```
bookServices  =  new  ExampleBookServices.CH14BookServicesCS();
bookServices.Url =
   "http://michaelk2/aspnetcookbook2cs/CSWebServices/CH14BookServicesCS.asmx";
```

Better still, by storing the URL in *web.config* and setting the URL property of the proxy class at runtime from the *web.config* setting, you can avoid changing the code whenever the URL changes.

Discussion

The ability to configure an application without having to recompile its code every time you make a change in the location of its resources can be a time-saver. Since the URL for a web service can change, code your application to set the URL at runtime.

Whenever you add a web reference to a project, Visual Studio 2005 adds an entry to the `<appSettings>` section of the *web.config* containing the URL of the web reference, as shown here for the two web references added for the recipes in this chapter:

```
<appSettings>
 <add key="ExampleWebServices.CH14QuickWebServiceVB2"
  value="http://localhost/…/CH14QuickWebServiceVB2.asmx"/>
  <add key="ExampleBookServices.CH14BookServicesVB"
    value="http://localhost/…/CH14BookServicesVB.asmx"/>
</appSettings>
```

```
<appSettings>
  <add key="ExampleWebServices.CH14QuickWebServiceCS2"
    value="http://localhost/…/CH14QuickWebServiceCS2.asmx"/>
  <add key="ExampleBookServices.CH14BookServicesCS"
    value="http://localhost/…/CH14BookServicesCS.asmx"/>
</appSettings>
```

By using the URL in *web.config* and setting the URL property of the web service at runtime, no code changes are required when the URL changes:

```
bookServices.Url = _
   ConfigurationManager.AppSettings.Item("ExampleBookServices.CH14BookServicesVB")
```

```
bookServices.Url =
   ConfigurationManager.AppSettings["ExampleBookServices.CH14BookServicesCS"];
```

PREV

Chapter 15. Dynamic Images

PREV

15.0 Introduction

The ability to draw or retrieve and display graphicimages on your web pages on the fly can add powerful functionality to an application. This is a nearly impossible task in classic ASP unless you use a third-party component. By contrast, the drawing library provided in the .NET Framework simplifies creating your own images when you need them. Indeed, it provides the ability to do almost anything you can imagine for image generation. The examples shown in this chapter show you how to:

- Draw button images on the fly using text generated during the running of your application
- Create bar charts on the fly
- Display images stored in a database
- Display thumbnails from full-sized images stored in a database

These represent a sampling of what you can do with the .NET drawing libraries and a little bit of custom code.

Recipe 15.2. Drawing Button Images on the Fly

Problem

You need to create a button image on the fly using text generated during the running of your application.

Solution

Create a web form that is responsible for creating the button image using the`System.Drawing` classes and then streaming the image to the `Response` object.

In the *.aspx* file, enter an `@ Page` directive, but omit any head or body tags. The`@ Page` directive links the ASP.NET page to the code-behind class that draws the image.

In the code-behind class for the page, use the .NET language of your choice to:

1. Import the `System.Drawing` and `System.Drawing.Imaging` namespaces.
2. Create a `makeButton` (or similarly named) method that creates a bitmap for a button using text generated during the running of the applicationfor example, text passed in the URL.
3. Create a `MemoryStream` object and save the bitmap in JPEG format (or other format) to the memory stream.
4. Write the resulting binary stream to `Response` object.

Examples 15-1, 15-2 through 15-3 show the *.aspx* file and VBand C# code-behind files for an application that creates a button image whose label is provided by the application user.

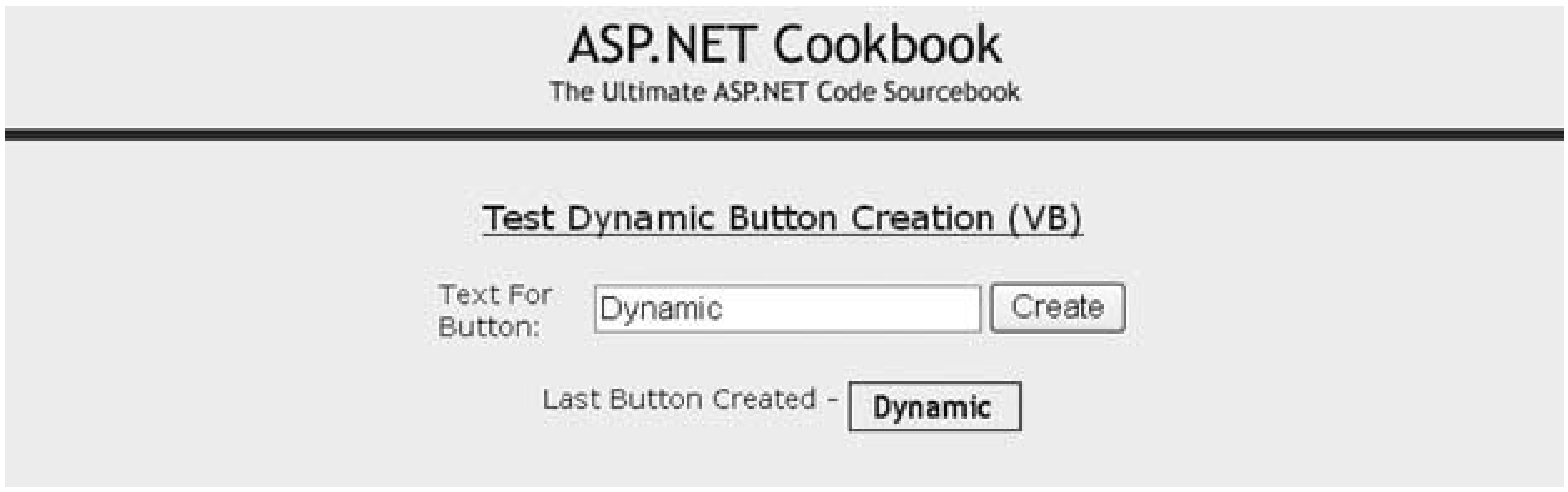
To use a dynamically generated image in your application, you need to set the`Src` attribute of the image tags for your button bitmaps to the URL of the ASP.NET page that creates the images, passing the image text in the URL.

In the *.aspx* file for the page, add an `img` tag for displaying the dynamically created image.

In the code-behind class for the page that uses the image, use the .NET language of your choice to set to the `Src` attribute of the image tag to the URL for the web form that will draw the image, passing the text it needs in the URL.

Examples 15-4, 15-5 through 15-6 show the *.aspx* file and VB and C# code-behind files for an application that uses the dynamic image generation.Figure 15-1 shows some typical output from the application.

Figure 15-1. Creating a button image on the fly


ASP.NET Cookbook

The Ultimate ASP.NET Code Sourcebook

Test Dynamic Button Creation (VB)

Text For Button: Dynamic

Create

Last Button Created - Dynamic

Discussion

Creating button images on the fly can be handy for two reasons. First, usingbutton images may help provide the look you want for your application. Second, generating them on the fly can avoid having to create and save a whole series of button images to the file system on the prospect they may be needed someday. For example, you might want to use images to improve the appearance of reports.

The approach we favor for generating images on the fly involves first drawing them and then streaming the images to the `Response` object. How does this work? The process begins when the browser first makes a request for a page to display. During the page rendering, whenever the browser encounters an image tag, it sends a request for that image to the server. The browser expects the server to stream the requested image back to the browser with the content type set to indicate an image of a certain type is being returnedfor example,`image/jpg`, indicating an image in JPEG format. Our approach does that, but with a twist. Instead of a static image, which is the norm, our approach returns an image that has been created on the fly on the server. The browser neither knows nor cares where the stream is coming from, which is why our approach works well.

Two web forms are used in our example that illustrates this solution. The first one renders no HTML but processes a user request for a dynamically createdbutton image. A second web form is used to display the requested image.

The *.aspx* file of the first form contains no head or body; it contains the`@ Page` directive to link the code-behind class for the page.

In the `Page_Load` event handler of the code-behind of the first page, the text for the image button passed in the URL is retrieved and passed to the`makeButton` method in the `buttonText` parameter to create the button image.

The `makeButton` method first creates the font that will be used to label the button image and then measures the space that will be required to display the text. Because we have no`Graphics` object in this scenario and a `Graphics` object cannot be created by itself (it must always be associated with a specific device context), we have to create a`Bitmap` solely for the purpose of creating the `Graphics` object. Here, a dummy one-pixel-by-one-pixel bitmap is created:

VB

```
bfont = New Font("Trebuchet MS", 10)
button = New Bitmap(1, 1, PixelFormat.Format32bppRgb)
g = Graphics.FromImage(button)
tSize = g.MeasureString(buttonText, bfont)
```

C#

```
bfont = new Font("Trebuchet MS", 10);
button = new Bitmap(1, 1, PixelFormat.Format32bppRgb);
g = Graphics.FromImage(button);
tSize = g.MeasureString(buttonText, bfont);
```

Next, we need to calculate the size of the image required to contain the text, allowing for some space around the text. The constants HORT_PAD and VERT_PAD are used to define the space at the ends of the text and above/below the text.

```
buttonWidth = CInt(Math.Ceiling(tSize.Width + (HORT_PAD * 2)))
buttonHeight = CInt(Math.Ceiling(tSize.Height + (VERT_PAD * 2)))
```

```
buttonWidth = Convert.ToInt32(Math.Ceiling(tSize.Width + (HORT_PAD * 2)));
buttonHeight = Convert.ToInt32(Math.Ceiling(tSize.Height + (VERT_PAD * 2)));
```

Now that we know how big to create the image, the Bitmap used for the button image can be created. The PixelFormat.Format32bppRgb defines the Bitmap to be 32 bits per pixel, using eight bits each for the red, green, and blue color components. For the image being created, the format is not that important. For more graphically appealing images, however, the format plays a significant role.

```
button = New Bitmap(buttonWidth, _
   buttonHeight, _
    PixelFormat.Format32bppRgb)
```

```
button = new Bitmap(buttonWidth,
   buttonHeight,
    PixelFormat.Format32bppRgb);
```

Next, the entire image is filled with a background color. This requires creating a brush with the desired color and calling the FillRectangle method with the full size of the image being created.

(Remember, the graphics coordinates always start at 0.) The FromHtml method of the ColorTranslator class is used to convert the HTML style color designation to the color required for the brush being created.

VB

```
g = Graphics.FromImage(button)
g.FillRectangle(New  SolidBrush(ColorTranslator.FromHtml("#F0F0F0")),  _
   0, _
   0, _
   buttonWidth - 1, _
   buttonHeight - 1)
```

C#

```
g = Graphics.FromImage(button);
g.FillRectangle(new  SolidBrush(ColorTranslator.FromHtml("#F0F0F0")),
   0,
   0,
   buttonWidth - 1,
   buttonHeight - 1);
```

When filling an image with a background color, be careful of the color choices you make, since browsers do not consistently display all colors. It is best to stick to the 216 colors of the web-safe palette.

After filling the button image background color, a border is drawn around the perimeter of the image This requires creating a pen with the desired color and width to do the drawing.

```
g.DrawRectangle(New  Pen(Color.Navy,  2),  _
   0, _
   0, _
   buttonWidth - 1, _
   buttonHeight - 1)
```

```
g.DrawRectangle(new  Pen(Color.Navy,  2),
   0,
   0,
   buttonWidth - 1,
   buttonHeight - 1);
```

Finally, the text is drawn in the center of the image button. The centering is accomplished by offsetting the upper-left corner of the text block by the same amount as the spacing that was allowe around the text.

VB

```
g.DrawString(buttonText, _
    bfont, _
    New SolidBrush(Color.Navy), _
    HORT_PAD, _
    VERT_PAD)
```

C#

```
g.DrawString(buttonText,
    bfont,
    new SolidBrush(Color.Navy),
    HORT_PAD,
    VERT_PAD);
```

When the button image bitmap is returned to the `Page_Load` method, we need to create a `MemoryStream` object and save the bitmap in JPEG format to the memory stream. (We chose the JPEG format because it happened to work well with the images in this example. Depending on the circumstances, you may need to use another format, such as GIF.)

```
ms = New MemoryStream
button.Save(ms, ImageFormat.Jpeg)
```

```
ms = new MemoryStream();
button.Save(ms, ImageFormat.Jpeg);
```

The final step in generating the button image is to write the binary stream to the `Response` object. This requires setting the `ContentType` property to match the format we saved the image to in the memory stream. In this example, the image was saved in JPEG format, so the `ContentType` must be set to `image/jpg`. This informs the browser that the stream being returned is an image of the proper type. We use the `BinaryWrite` method to write the image to the `Response` object.

```
Response.ContentType = "image/jpg"
Response.BinaryWrite(ms.ToArray())
```

```
Response.ContentType = "image/jpg";
 Response.BinaryWrite(ms.ToArray());
```

Using our example web form that dynamically creates button images requires setting the `Src` attribute of the image tag to the URL for the web form just described, passing the text for the image in the URL. A sample URL is shown here:

```
src="CH15CreateButtonVB.aspx?ButtonText=Dynamic"
```

In our example, the `Src` attribute of the `img` tag is set in the create button click event of the test page code-behind.

> Dynamically generating images can be resource intensive. To improve your application's performance, the generated images should be cached. Recipe 16.2 provides an example of how to cache the results as a function of the data passed in the `QueryString`.

See Also

Recipe 16.2 and MSDN Help for more on the .NET drawing libraries

Example 15-1. Create images dynamically (.aspx)

```
<%@ Page Language="VB" AutoEventWireup="false"
  CodeFile="CH15CreateButtonVB.aspx.vb"
  Inherits="ASPNetCookbook.VBExamples.CH15CreateButtonVB"  %>
```

Example 15-2. Create images dynamically code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Drawing
Imports System.Drawing.Imaging
Imports System.IO

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code-behind for
  '''  CH15CreateButtonVB.aspx
  ''' </summary>
  Partial Class CH15CreateButtonVB
```

```
Inherits System.Web.UI.Page

'constants used to create URLs to this page
Public Const PAGE_NAME As String = "CH15CreateButtonVB.aspx"
Public Const QS_BUTTON_TEXT As String = "ButtonText"

'''*************************************************************************
''' <summary>
''' This routine provides the event handler for the page load event. It
''' is responsible for initializing the controls on the page.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
 Dim buttonText As String
 Dim button As Bitmap
 Dim ms As MemoryStream
 'get button text from the query string and create the image
 buttonText = Request.QueryString(QS_BUTTON_TEXT)
 button = makeButton(buttonText)

 'write image to response object
 ms = New MemoryStream
 button.Save(ms, ImageFormat.Jpeg)
 Response.ContentType = "image/jpg"
 Response.BinaryWrite(ms.ToArray( ))
End Sub 'Page_Load

'''*************************************************************************
''' <summary>
''' This routine creates a button with the passed text.
''' </summary>
'''
''' <param name="buttonText">
''' Set to the text to place on the button
''' </param>
'''
''' <returns>Bitmap of button that was created</returns>
Private Function makeButton(ByVal buttonText As String) As Bitmap
 'define the space around the text on the button
 Const HORT_PAD As Integer = 10
 Const VERT_PAD As Integer = 2

 Dim button As Bitmap
 Dim g As Graphics
 Dim bfont As Font
 Dim tSize As SizeF
 Dim buttonHeight As Integer
 Dim buttonWidth As Integer
```

```
    'create the font that will used then create a dummy button to get
    'a graphics object that provides the ability to measure the height
    'and width required to display the passed string
    bfont = New Font("Trebuchet MS", 10)
    button = New Bitmap(1, 1, PixelFormat.Format32bppRgb)
    g = Graphics.FromImage(button)
    tSize = g.MeasureString(buttonText, bfont)

    'calculate the size of button required to display the text adding
    'some space around the text
    buttonWidth = CInt(Math.Ceiling(tSize.Width + (HORT_PAD * 2)))
    buttonHeight = CInt(Math.Ceiling(tSize.Height + (VERT_PAD * 2)))

    'create a new button using the calculated size
    button = New Bitmap(buttonWidth, _
      buttonHeight, _
      PixelFormat.Format32bppRgb)
    'fill the button area
    g = Graphics.FromImage(button)
    g.FillRectangle(New SolidBrush(ColorTranslator.FromHtml("#F0F0F0")), _
     0, _
     0, _
     buttonWidth - 1, _
     buttonHeight - 1)

    'draw a rectangle around the button perimeter using a pen width of 2
    g.DrawRectangle(New Pen(Color.Navy, 2), _
     0, _
     0, _
     buttonWidth - 1, _
     buttonHeight - 1)

    'draw the text on the button (centered)
    g.DrawString(buttonText, _
      bfont, _
      New SolidBrush(Color.Navy), _
      HORT_PAD, _
      VERT_PAD)
    g.Dispose( )
    Return (button)
   End Function 'makeButton
 End Class 'CH15CreateButtonVB
End Namespace
```

Example 15-3. Create images dynamically code-behind (.cs)

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH15CreateButtonCS.aspx
 /// </summary>
public partial class CH15CreateButtonCS : System.Web.UI.Page
{
 // constants used to create URLs to this page
 public const String PAGE_NAME = "CH15CreateButtonCS.aspx";
 public const String QS_BUTTON_TEXT = "ButtonText";

 ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
  String buttonText = null;
  Bitmap button = null;
  MemoryStream ms = null;

  // get button text from the query string and create image
  buttonText = Request.QueryString[QS_BUTTON_TEXT];
  button = makeButton(buttonText);

  // write image to response object
  ms = new MemoryStream( );
  button.Save(ms, ImageFormat.Jpeg);
  Response.ContentType = "image/jpg";
  Response.BinaryWrite(ms.ToArray( ));
  } // Page_Load

  ///*********************************************************************
  /// <summary>
  /// This routine creates a button with the passed text.
  /// </summary>
  ///
  /// <param name="buttonText">
  /// Set to the text to place on the button
  /// </param>
  private Bitmap makeButton(String buttonText)
  {
   // define the space around the text on the button
```

```
const int HORT_PAD = 10;
const int VERT_PAD = 2;

Bitmap button = null;
Graphics g = null;
Font bfont = null;
SizeF tSize;
int buttonHeight;
int buttonWidth;

// create the font that will used then create a dummy button to get
// a graphics object that provides the ability to measure the height
// and width required to display the passed string
bfont = new Font("Trebuchet MS", 10);
button = new Bitmap(1, 1, PixelFormat.Format32bppRgb);
g = Graphics.FromImage(button);
tSize = g.MeasureString(buttonText, bfont);

// calculate the size of button required to display the text adding
// some space around the text
buttonWidth = Convert.ToInt32(Math.Ceiling(tSize.Width +
       (HORT_PAD * 2)));
buttonHeight = Convert.ToInt32(Math.Ceiling(tSize.Height +
   (VERT_PAD * 2)));

// create a new button using the calculated size
button = new Bitmap(buttonWidth,
  buttonHeight,
  PixelFormat.Format32bppRgb);

// fill the button area
g = Graphics.FromImage(button);
g.FillRectangle(new SolidBrush(ColorTranslator.FromHtml("#F0F0F0")),
 0,
 0,
 buttonWidth - 1,
 buttonHeight - 1);

// draw a rectangle around the button perimeter using a pen width of 2
g.DrawRectangle(new Pen(Color.Navy, 2),
 0,
 0,
 buttonWidth - 1,
 buttonHeight - 1);

// draw the text on the button (centered)
g.DrawString(buttonText,
  bfont,
  new SolidBrush(Color.Navy),
  HORT_PAD,
  VERT_PAD);
g.Dispose( );
```

```
   return (button);
  } // makeButton
 } // CH15CreateButtonCS
}
```

Example 15-4. Using the dynamically created images (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH15TestCreateButtonVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH15TestCreateButtonVB"
 Title="Test Dynamic Button Creation" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Test Dynamic Button Creation (VB)
 </div>
 <table width="50%" align="center" border="0">
  <tr>
   <td class="labelText">Text For Button: </td>
   <td>
    <asp:TextBox ID="txtButtonText" Runat="server" />
   </td>
   <td>
    <input id="btnCreate" runat="server"
      type="button"
      value="Create"
      name="btnCreate"
      onserverclick="btnCreate_ServerClick"/>
   </td>
  </tr>
  <tr>
   <td id="tdCreatedButton" runat="server"
       colspan="3"
       align="center"
       class="labelText">
    <br />
    Last Button Created - <img id="imgButton" runat="server"
          border="0" src=""
          align="middle" alt="button"/>
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 15-5. Using the dynamically created images code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH15TestCreateButtonVB.aspx
 ''' </summary>
 Partial  Class  CH15TestCreateButtonVB
  Inherits System.Web.UI.Page
   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   If (Not Page.IsPostBack) Then
    'make image button table cell invisible initially
    tdCreatedButton.Visible = False
   End If
  End Sub 'Page_Load

   '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the create button click
  ''' event. It is responsible for initializing the source property of the
  ''' image button to the URL of the dynamic button creation page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnCreate_ServerClick(ByVal sender As Object, _
                     ByVal e As System.EventArgs) _
   'update the image tag with the URL to the page that will
   'create the button and with the button text in the URL
   imgButton.Src = "CH15CreateButtonVB.aspx?ButtonText=" & _
    txtButtonText.Text
   tdCreatedButton.Visible = True
  End Sub 'btnCreate_Click
 End  Class  'CH15TestCreateButtonVB
End Namespace
```

Example 15-6. Using the dynamically created images code-behind (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH15TestCreateButtonCS.aspx
 /// </summary>
 public partial class CH15TestCreateButtonCS : System.Web.UI.Page
 {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   if (!Page.IsPostBack)
   {
    // make image button table cell invisible initially
    tdCreatedButton.Visible = false;
   }
  } // Page_Load

  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the create button click
  /// event. It is responsible for initializing the source property of the
  /// image button to the URL of the dynamic button creation page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnCreate_ServerClick(Object sender,
           System.EventArgs e)
  {
   // update the image tag with the URL to the aspx page that will
   // create the button and with the button text in the URL
   imgButton.Src = "CH15CreateButtonCS.aspx?ButtonText=" +
    txtButtonText.Text;
   tdCreatedButton.Visible = true;
  } // btnCreate_ServerClick
 } // CH15TestCreateButtonCS
}
```

PREV

Recipe 15.3. Creating Bar Charts on the Fly

Problem

You want to create a bar chart on the fly without having to resort to a commercial package.

Solution

Use a combination of data binding with a `Repeater` control and the well-known HTML trick of stretching an image to create the bars.

In the *.aspx* file, add a `Repeater` control with an `ItemTemplate` .

In the code-behind class for the page, use the .NET language of your choice to:

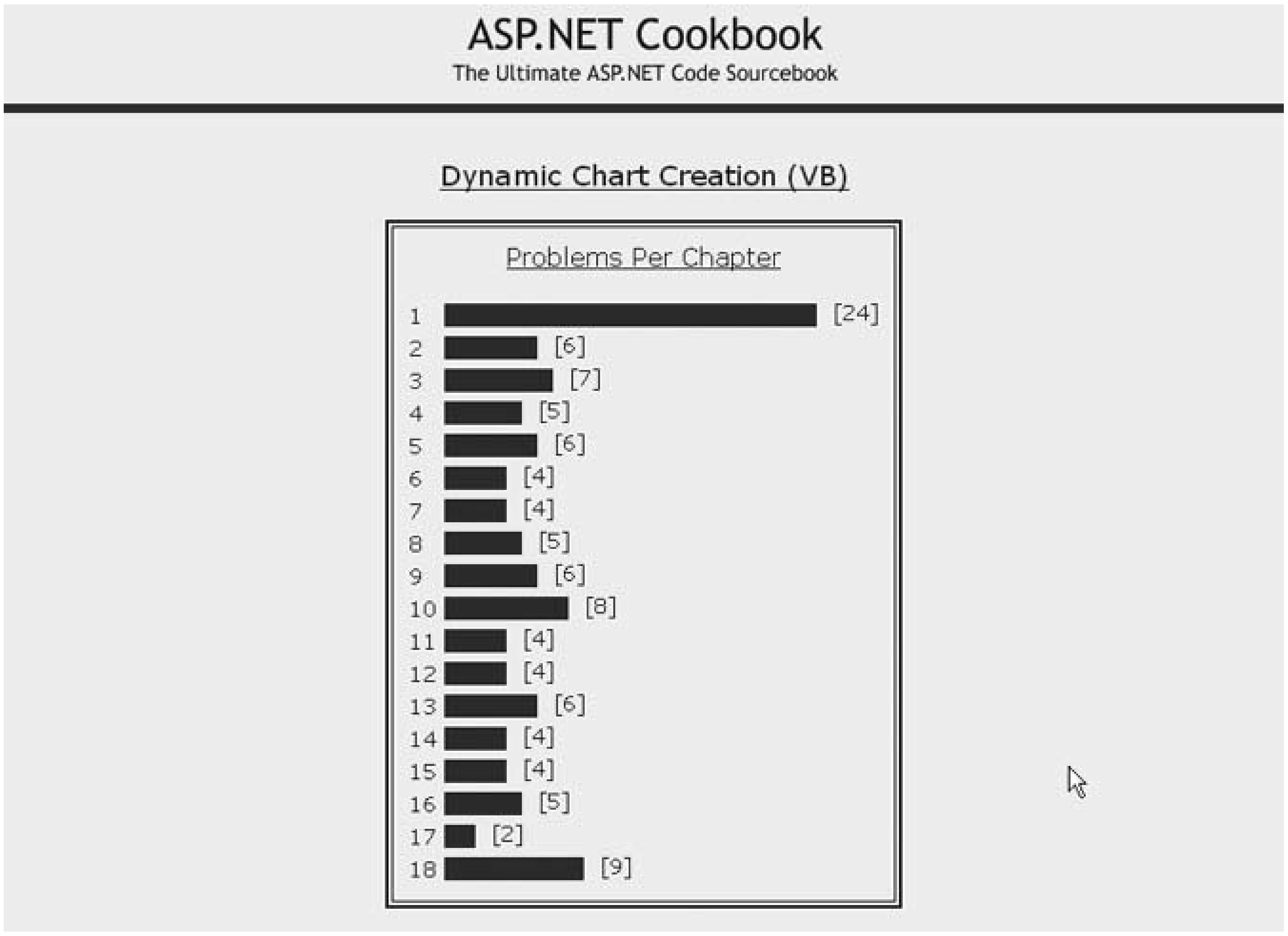
1. Assign the data source to the `Repeater` control and bind it.
2. In the `ItemDataBound` event handler called for each item in the `Repeater` , set the width of the bar in the passed `Repeater` row.

Figure 15-2 shows some typical output. Examples 15-7 , 15-8 through 15-9 show the *.aspx* file and VB and C# code-behind files for an application that implements this solution.

Discussion

This recipe provides a simple approach that combines data binding and HTML tricks to create a bar graph with little coding and without the need to purchase any additional components. By using more complex HTML, you can add more labels and other enhancements to this recipe, which may make it more useful for your situation.

Figure 15-2. Creating a bar chart output dynamically


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Dynamic Chart Creation (VB)
Problems Per Chapter

1	[24]
2	[6]
3	[7]
4	[5]
5	[6]
6	[4]
7	[4]
8	[5]
9	[6]
10	[8]
11	[4]
12	[4]
13	[6]
14	[4]
15	[4]
16	[5]
17	[2]
18	[9]

The example we use to illustrate this solution generates a bar chart from chapter and problem data i a database. (The source of the data is not that important; the technique for generating the graph on the fly is the focus of this recipe.) The bar chart is created from an HTML table, with the top row use to label the chart.

This recipe advocates using a `Repeater` control to generate the rows in a table that represent the bars on the chart. The rows generated by the `Repeater` are defined in the `ItemTemplate` element, which in our example contains two columns. The first column is used to output the chapter number. In our example, the chapter number is obtained by binding the cell text to the `Chapter` column in the data source.

The second column contains the bar representing the number of problems in the chapter. The bar is created by using an HTML image tag with the source set to a one-pixel-by-one-pixel image. The height and width attributes of the image "stretch" the image to the size of the bar needed to represent the number of problems in the chapter. In our example, the height is set to a fixed value o 15 pixels, but the width is adjusted to represent the number of problems in the chapter. The width is adjusted in the code-behind and is discussed later.

The second column also contains a label to indicate the number of problems in the chapter. The number of problems in a chapter is obtained by binding the cell text to the `ProblemCount` column in the data source. This label is placed at the end of thebar with a non-breaking space (`&nbsp;`) to separate the label from the end of the bar.

The Page_Load method in the example code-behind reads the data from the database and then binds the data to the Repeater control on the page.

The code-behind class also implements the ItemDataBound event handler to provide the ability to adjust the width of the image used for the bar. The ItemDataBound event executes once per row in the Repeater as the row is data bound. In this event, we need to get a reference to the HTML image in the row using the FindControl method of the row and set the width of the image to reflect the number of problems in the chapter represented by the row.

If this recipe does not provide the richness you need for your chart, you can create an image using the concepts presented in Recipe 15.1. The System.Drawing classes provide all of the functionality to create sophisticated charts using the GDI+ Library. They do require more coding, however, as you must build your graphs from the ground up using the basic ingredients of pen, brush, point, rectangle, etc.

See Also

Recipe 15.1 and MSDN Help for more information on the System.Drawing class

Example 15-7. Creating a bar chart dynamically (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH15CreateChartVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH15CreateChartVB"
 Title="Create Chart" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Dynamic Chart Creation (VB)
 </div>
 <table align="center" border="2" cellpadding="10">
  <tr>
   <td>
     <table cellpadding="0" cellspacing="0" border="0">
    <tr>
     <td align="center" colspan="2" class="subHeading">
    Problems Per Chapter<br /><br /></td>
    </tr>
    <asp:Repeater ID="repChartBar" Runat="server"
     OnItemDataBound="repChartBar_ItemDataBound">
    <ItemTemplate>
    <tr>
     <td class="labelText">
      <%#Eval("Chapter")%>&nbsp;
     </td>
     <td class="labelText">
      <img id="imgChartBar" runat="server"
       src="images/blueSpacer.gif" border="0"
       height="15" align="middle" alt="bar"/>
      &nbsp;[<%#Eval("ProblemCount")%>]
     </td>
    </tr>
    </ItemTemplate>
    </asp:Repeater>
    </table>
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 15-8. Create bar chart dynamically code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Configuration.ConfigurationManager
Imports System.Data
Imports System.Data.Common
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH15CreateChartVB.aspx
 ''' </summary>
 Partial Class CH15CreateChartVB
  Inherits System.Web.UI.Page
  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim dbConn As OleDbConnection = Nothing
   Dim dc As OleDbCommand
   Dim dr As OleDbDataReader
   Dim strConnection As String
   Dim strSQL As String

   If (Not Page.IsPostBack) Then
    Try
    'get the connection string from web.config and open a connection
    'to the database
    strConnection = _
    ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDb.OleDbConnection(strConnection)
    dbConn.Open( )

    'build the query string and get the data from the database
    strSQL = "SELECT DISTINCT ChapterID AS Chapter, " & _
     "count(*) AS ProblemCount " & _
     "FROM Problem " &_
     "GROUP BY ChapterID"

    dc = New OleDbCommand(strSQL, dbConn)
    dr = dc.ExecuteReader( )

    'set the source of the data for the repeater control and bind it
    repChartBar.DataSource = dr
    repChartBar.DataBind( )

    Finally
    'clean up
    If (Not IsNothing(dbConn)) Then
    dbConn.Close( )
    End If
    End Try
  End If
End Sub 'Page_Load

 '''*************************************************************************
''' <summary>
''' This routine is the event handler that is called for each item in the
''' repeater after a data bind occurs. It is responsible for setting the
''' width of the bar in the passed repeater row to reflect the number of
''' problems in the chapter the row represents
''' </summary>
'''
''' <param name="sender"></param>
''' <param name="e"></param>
''' <remarks></remarks>
Protected Sub repChartBar_ItemDataBound(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.RepeaterItemEventArgs)
  Dim img As System.Web.UI.HtmlControls.HtmlImage

  'get a reference to the image used for the bar in the row
  img = CType(e.Item.FindControl("imgChartBar"), _
    System.Web.UI.HtmlControls.HtmlImage)

  'set the width to the number of problems in the chapter for this row
  'multiplied by a constant to stretch the bar a bit more
  img.Width = _
    CInt(CType(e.Item.DataItem, DbDataRecord)("ProblemCount")) * 10
  End Sub 'repChartBar_ItemDataBound
 End Class 'CH15CreateChartVB
End Namespace
```

Example 15-9. Creating a bar chart dynamically code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.Common;
using System.Data.OleDb;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH15CreateChartCS.aspx
 /// </summary>
 public partial class CH15CreateChartCS : System.Web.UI.Page
 {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   OleDbConnection dbConn = null;
   OleDbCommand dc = null;
   OleDbDataReader dr = null;
   string strConnection = null;
   String strSQL = null;

   if (!Page.IsPostBack)
   {
    try
    {
    // get the connection string from web.config and open a connection
    // to the database
    strConnection = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(strConnection);
    dbConn.Open( );
```

```
     // build the query string and get the data from the database
     strSQL = "SELECT DISTINCT ChapterID AS Chapter, " +
      "count(*) AS ProblemCount " +
      "FROM Problem " +
      "GROUP BY ChapterID";
     dc = new OleDbCommand(strSQL, dbConn);
     dr = dc.ExecuteReader( );

     // set the source of the data for the repeater control and bind it
     repChartBar.DataSource = dr;
     repChartBar.DataBind( );
     }

     finally
     {
     // clean up
     if (dbConn != null)
     {
     dbConn.Close( );
     }
     }
    }
   } // Page_Load

   ///**************************************************************************
   /// <summary>
   /// This routine is the event handler that is called for each item in the
   /// repeater after a data bind occurs. It is responsible for setting the
   /// width of the bar in the passed repeater row to reflect the number of
   /// problems in the chapter the row represents.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
   protected void repChartBar_ItemDataBound(Object sender,
        System.Web.UI.WebControls.RepeaterItemEventArgs e)
   {
    System.Web.UI.HtmlControls.HtmlImage img = null;

    // get a reference to the image used for the bar in the row
    img = (System.Web.UI.HtmlControls.HtmlImage)
       (e.Item.FindControl("imgChartBar"));

    // set the width to the number of problems in the chapter for this row
    // multiplied by a constant to stretch the bar a bit more
    img.Width =
     (int)(((DbDataRecord)(e.Item.DataItem))["ProblemCount"]) * 10;
   } // repChartBar_ItemDataBound
  } // CH15CreateChartCS
 }
```

PREV

Recipe 15.4. Displaying Images Stored in a Database

Problem

Your application stores images in a database that you want to display on a web form.

Solution

Create a web form that reads the image data from the database and streams the image to the `Response` object.

In the *.aspx* file, enter an @ `Page` directive and omit any head or body tags to link the*.aspx* page to the code-behind class that retrieves and displays the images:

1. Read the image ID that is generated by the running applicationfor example, the image ID passed in the URL for accessing the web form.
2. Open a connection to the database that contains the images.
3. Build a query string, and read the byte array of the desired image from the database.
4. Set the content type for the image and write it to the`Response` object.

Examples 15-10 , 15-11 through 15-12 show the *.aspx* file and VB and C# code-behind files for an application that implements the image-building portion of the solution.

To use this dynamic image generation technique in your application, set the`src` attribute of the image tags used to display the images to the URL of the ASP.NET page that reads the images from the database, passing the image ID in the URL.

In the *.aspx* file for the page, add an `img` tag for displaying the image.

In the code-behind class for the page that uses the image, use the .NET language of your choice to set the `src` attribute of the image tag to the URL for the web form just described, passing the ID of the image in the URL.

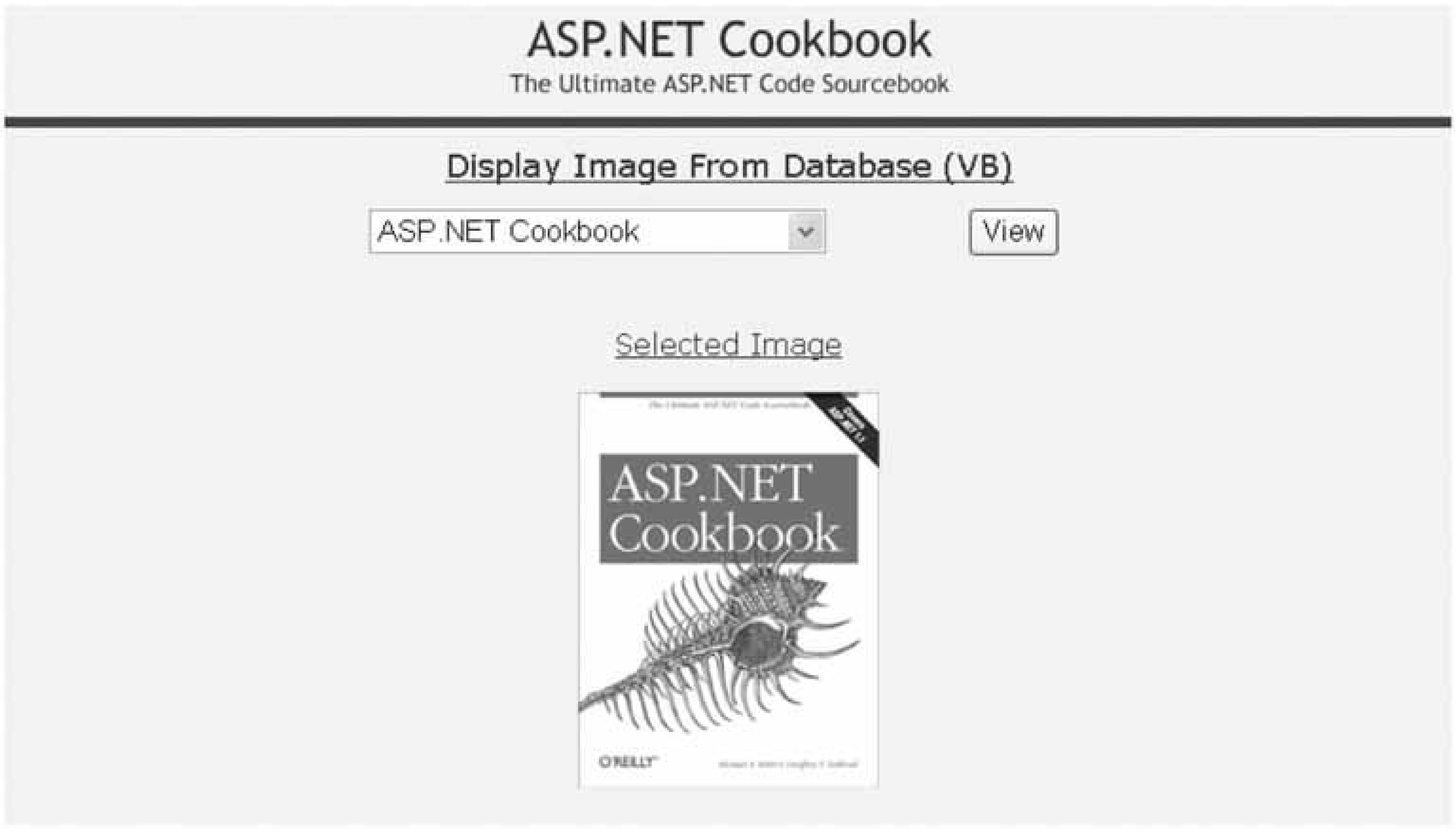
Examples 15-13 , 15-14 through 15-15 show the *.aspx* file and VB and C# code-behind files for an application that uses the dynamic image generation. Figure 15-3shows some typical output from the application.

Discussion

If you have images in a database that you want to display on a web form, chances are you've considered using an image tag to display them. Nevertheless, you may be searching for a practical way to set the image tag's `src` attribute and move the image data to the browser while maintaining the maximum flexibility in selecting the images you need.

The solution we favor involves creating a web form that reads an image from a database and stream it to the `Response` object. A convenient way to specify the image to read from the database is to include an image ID in the URL used to call the web form that retrieves and returns the image to the browser.

Figure 15-3. Displaying images from a database



Our example that illustrates this solution consists of two web forms. The first web form renders no HTML but instead processes the request for reading an image from the database. The second web form is used to demonstrate displaying an image from the database.

The *.aspx* file of the first form contains no head or body; it simply contains the `@Page` directive to link the page to its code-behind class.

The `Page_Load` method of the code-behind performs the following four steps to retrieve the requested image from the database and send it to the browser:

1. Retrieve the ID of the requested image from the URL.
2. Read the byte array of the image from the database.
3. Set the `ContentType` of the type of image stored in the database (GIF in this example).

4. Write the byte array of the image to the `Response` object.

To use the ASP.NET page to retrieve images from the database, we need to set the`src` attribute of an image tag to the name of the page just described, passing the ID of the desired image in the URL A sample URL is shown here:

```
src="CH15ImageFromDatabaseVB.aspx?ImageID=13"
```

In our example, the `src` attribute of an `img` tag is set in the view image button click event of the test page code-behind.

The performance of this solution can be significantly improved by caching the images retrieved from the database instead of retrieving them for each request. Refer to Recipe 16.3 for an example of how to cache the results as a function of the data passed in the`QueryString` .

An `HttpHandler` can be used to implement the same functionality described in this recipe. Refer to Recipe 20.1 for an example of retrievingimages from a database using an `HttpHandler` .

Images can be stored in a database using the technique described in Recipe 18.4.

See Also

Recipes 16.2, 18.4, and 20.1

Example 15-10. Reading images from a database (.aspx)

```
<%@ Page Language="VB" AutoEventWireup="false"
 CodeFile="CH15ImageFromDatabaseVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH15ImageFromDatabaseVB"   %>
```

Example 15-11. Reading images from a database code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Configuration.ConfigurationManager
Imports System.Data
Imports System.Data.Common
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
```

```
''' <summary>
''' This class provides the code-behind for
'''  CH15ImageFromDatabaseVB.aspx
''' </summary>
Partial  Class  CH15ImageFromDatabaseVB
 Inherits System.Web.UI.Page

 'constants used to create URLs to this page
 Public  Const  PAGE_NAME  As  String  =  "CH15ImageFromDatabaseVB.aspx"
 Public Const QS_IMAGE_ID As String = "ImageID"
 '''**************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Private Sub Page_Load(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Me.Load
  Dim dbConn As OleDbConnection = Nothing
  Dim dCmd As OleDbCommand
  Dim imageData( ) As Byte
  Dim strConnection As String
  Dim strSQL As String
  Dim imageID As String

  If (Not Page.IsPostBack) Then
   Try
   'get the ID of the image to retrieve from the database
   imageID = Request.QueryString(QS_IMAGE_ID)

   'get the connection string from web.config and open a connection
   'to the database
   strConnection = _
   ConnectionStrings("dbConnectionString").ConnectionString
   dbConn = New OleDb.OleDbConnection(strConnection)
   dbConn.Open( )

   'build the query string and get the data from the database
   strSQL = "SELECT ImageData " &_
    "FROM BookImage " &_
    "WHERE BookImageID=?"
   dCmd = New OleDbCommand(strSQL, dbConn)
   dCmd.Parameters.Add(New OleDbParameter("BookImageID", imageID))
   imageData = CType(dCmd.ExecuteScalar( ), Byte( ))

   'set the content type for the image and write it to the response
   Response.ContentType = "image/gif"
   Response.BinaryWrite(imageData)

   Finally
```

```
    'clean up
    If (Not IsNothing(dbConn)) Then
    dbConn.Close( )
    End If
    End Try
   End If
  End Sub 'Page_Load
 End  Class  'CH15ImageFromDatabaseVB
End Namespace
```

Example 15-12. Reading images from a database code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.Common;
using System.Data.OleDb;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH15ImageFromDatabaseCS.aspx
 /// </summary>
 public  partial  class  CH15ImageFromDatabaseCS  :  System.Web.UI.Page
 {
  // constants used to create URLs to this page
   public  const  String  PAGE_NAME  =  "CH15ImageFromDatabaseCS.aspx";
  public const String QS_IMAGE_ID = "ImageID";

   ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   OleDbConnection dbConn = null;
   OleDbCommand dCmd = null;
   byte[] imageData = null;
   String strConnection = null;
   String strSQL = null;
   String imageID = null;
   if (!Page.IsPostBack)
   {
    try
    {
    // get the ID of the image to retrieve from the database
    imageID = Request.QueryString[QS_IMAGE_ID];

    // get the connection string from web.config and open a connection
    // to the database
    strConnection = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(strConnection);
    dbConn.Open( );

    // build the query string and get the data from the database
    strSQL = "SELECT ImageData " +
     "FROM BookImage " +
     "WHERE BookImageID=?";
    dCmd = new OleDbCommand(strSQL, dbConn);
    dCmd.Parameters.Add(new OleDbParameter("BookImageID", imageID));
    imageData = (byte[])(dCmd.ExecuteScalar( ));

    // set the content type for the image and write it to the response
    Response.ContentType = "image/gif";
    Response.BinaryWrite(imageData);
   }
    finally
    {
    // clean up
    if (dbConn != null)
    {
    dbConn.Close( );
    }
    }
   }
  } // Page_Load
 } //  CH15ImageFromDatabaseCS
}
```

Example 15-13. Displaying images from a database (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH15TestImageFromDatabaseVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH15TestImageFromDatabaseVB"
 Title="Test Image From Database" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Display Image From Database (VB)
 </div>
 <table align="center" width="50%" border="0">
  <tr>
   <td>
    <asp:DropDownList ID="ddImages" Runat="server" />
   </td>
   <td>
    <input id="btnViewImage" runat="server"
    type="button"
    value="View"
    onserverclick="btnViewImage_ServerClick"/>
   </td>
  </tr>
  <tr>
   <td id="tdSelectedImage" runat="server"
     colspan="2" align="center" class="subHeading">
    <br /><br />
    Selected Image<br /><br />
    <img id="imgBook" runat="server" border="0" src="" alt="book"/>
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 15-14. Displaying images from a database code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System.Configuration
Imports System.Data
Imports System.Data.Common
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH15TestImageFromDatabaseVB.aspx
 ''' </summary>
 Partial  Class  CH15TestImageFromDatabaseVB
  Inherits System.Web.UI.Page
  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Dim dbConn As OleDbConnection = Nothing
   Dim dc As OleDbCommand
   Dim dr As OleDbDataReader
   Dim strConnection As String
   Dim strSQL As String

   If (Not Page.IsPostBack) Then
    'initially hide the selected image since one is not selected
    tdSelectedImage.Visible = False

    Try
    'get the connection string from web.config and open a connection
    'to the database
    strConnection = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDb.OleDbConnection(strConnection)
    dbConn.Open( )

    'build the query string and get the data from the database
    strSQL = "SELECT BookImageID, Title " &_
     "FROM BookImage"
    dc = New OleDbCommand(strSQL, dbConn)
    dr = dc.ExecuteReader( )

    'set the source of the data for the repeater control and bind it
    ddImages.DataSource = dr
    ddImages.DataTextField = "Title"
    ddImages.DataValueField = "BookImageID"
    ddImages.DataBind( )
    Finally
    'clean up
    If (Not IsNothing(dbConn)) Then
    dbConn.Close( )
    End If
    End Try
   End If
  End Sub 'Page_Load

  '''*************************************************************************
```

```
  ''' <summary>
  ''' This routine provides the event handler for the view image click event.
  ''' It is responsible for setting the src attibute of the imgBook tag to
  ''' the page that will retrieve the image data from the database and
  ''' stream to the browser.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnViewImage_ServerClick(ByVal sender As Object, _
             ByVal e As System.EventArgs)
   'set the source for the selected image tag
   imgBook.Src = "CH15ImageFromDatabaseVB.aspx?ImageID=" &_
    ddImages.SelectedItem.Value.ToString( )
    'make the selected image visible
   tdSelectedImage.Visible = True
  End Sub 'btnViewImage_ServerClick
 End  Class  'CH15TestImageFromDatabaseVB
End Namespace
```

Example 15-15. Displaying images from a database code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH15TestImageFromDatabaseCS.aspx
 /// </summary>
 public  partial  class  CH15TestImageFromDatabaseCS  :  System.Web.UI.Page
 {
   ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   OleDbConnection dbConn = null;
   OleDbCommand dc = null;
```

```
    OleDbDataReader dr = null;
    String strConnection = null;
    String strSQL = null;

    if (!Page.IsPostBack)
    {
     // initially hide the selected image since one is not selected
     tdSelectedImage.Visible = false;

     try
     {
     // get the connection string from web.config and open a connection
     // to the database
     strConnection = ConfigurationManager.
     ConnectionStrings["dbConnectionString"].ConnectionString;
     dbConn = new OleDbConnection(strConnection);
     dbConn.Open( );

     // build the query string and get the data from the database
     strSQL = "SELECT BookImageID, Title " +
      "FROM BookImage";
     dc = new OleDbCommand(strSQL, dbConn);
     dr = dc.ExecuteReader( );

     // set the source of the data for the repeater control and bind it
     ddImages.DataSource = dr;
     ddImages.DataTextField = "Title";
     ddImages.DataValueField = "BookImageID";
     ddImages.DataBind( );
     }

     finally
     {
     // clean up
     if (dbConn != null)
     {
     dbConn.Close( );
     }
     }
    }
   } // Page_Load

   ///***************************************************************************
   /// <summary>
   /// This routine provides the event handler for the view image click
   /// event. It is responsible for setting the src attibute of the imgBook
   /// tag to the page that will retrieve the image data from the database
   /// and stream it to the browser.
   /// </summary>
   ///
   /// <param name="sender">Set to the sender of the event</param>
   /// <param name="e">Set to the event arguments</param>
```

```
  protected void btnViewImage_ServerClick(Object sender,
              System.EventArgs e)
  {
   // set the source for the selected image tag
   imgBook.Src = "CH15ImageFromDatabaseCS.aspx?ImageID=" +
    ddImages.SelectedItem.Value.ToString( );

   // make the selected image visible
   tdSelectedImage.Visible = true;
  } // btnViewImage_ServerClick
 } // CH15TestImageFromDatabaseCS
}
```

Recipe 15.5. Displaying Thumbnail Images

Problem

You want to display a page of images stored in your database and scaled on the fly to thumbnail format.

Solution

Implement the first of the two ASP.NET pages described in Recipe 15.3, changing the Page_Load method in the code-behind class to scale the full-sized image retrieved from the database to the appropriate size for a thumbnail presentation.

In the Page_Load method of the code-behind class for the page, use the .NET language of your choice to:

1. Create a System.Drawing.Image object from the byte array retrieved from the database.
2. Use a constant to define the height of the thumbnail and calculate the width to maintain the aspect ratio of the image.
3. Use the GetThumbnailImage method of the Image object to scale the image to the desired size.
4. Load the thumbnail image into a MemoryStream and write it to the Response object.

Examples 15-16 and 15-17 show the VB and C# code-behind class for our example that illustrates this solution. (See the *CH15ImageFromDatabaseVB .aspx* file and VB and C# code-behind files in Recipe 15.3 for our starting point.)

To display the thumbnails, create another ASP.NET page, add a DataList control with image tags in the ItemTemplate, and use data binding to set the src attributes of the image tags.

In the *.aspx* file for the page:

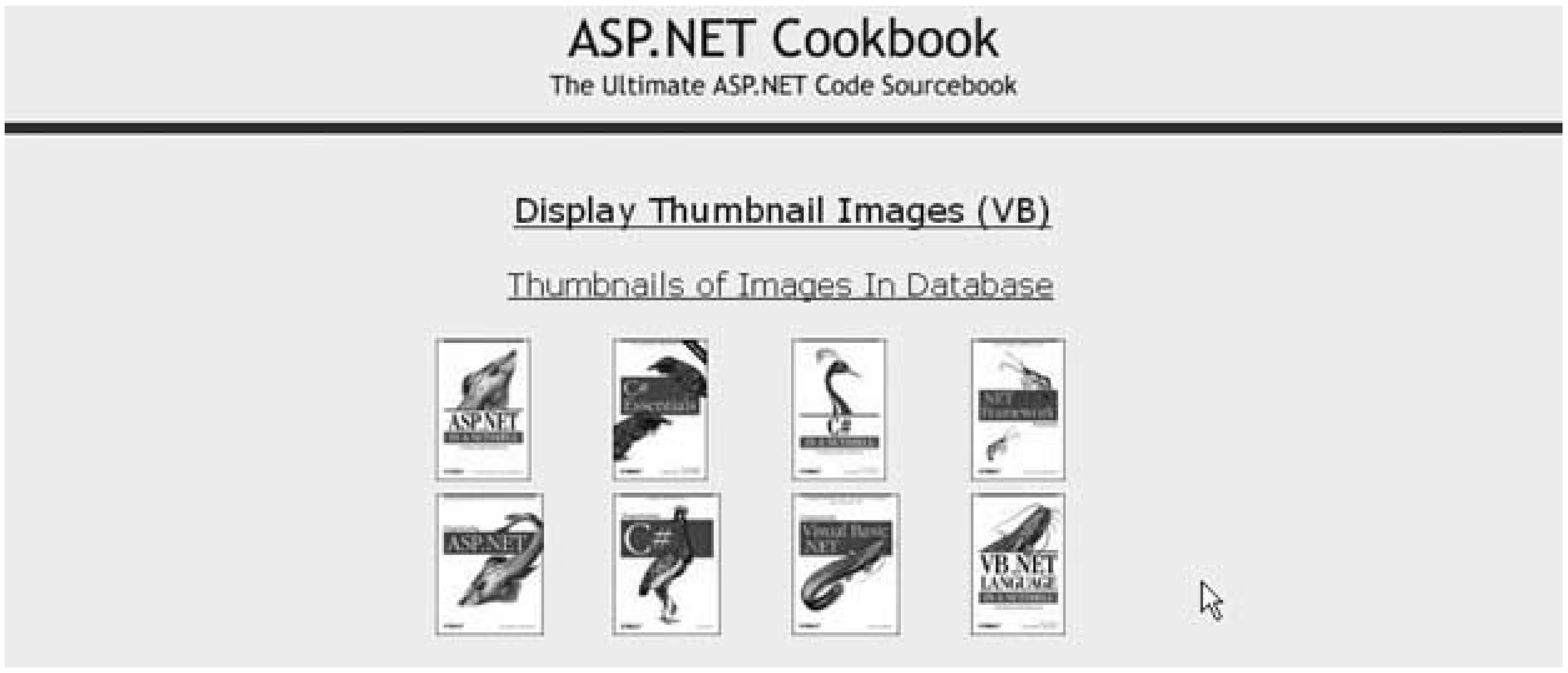
1. Use a DataList control to provide the ability to generate a list using data binding.
2. Use a HeaderTemplate to label the table of images.
3. Use an ItemTemplate to define an image that is displayed in the DataList.

In the code-behind class for the page, use the .NET language of your choice to:

1. Open a connection to the database.
2. Read the list of images from the database.
3. Assign the data source to the `DataList` control and bind it.
4. Set the `src` attribute of the image tags used to display the thumbnail images in the `ItemDataBound` event of the `DataList`.

Examples 15-18, 15-19 through 15-20 show the *.aspx* file and VB and C# code-behind files for the application that uses the dynamically generated images. The output produced by the page is shown in Figure 15-4.

Figure 15-4. Displaying thumbnails output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Display Thumbnail Images (VB)
Thumbnails of Images In Database

Discussion

The rationale for this recipe is similar to that of Recipe 15.3. You need a convenient way to display images from a database, in this case a page of thumbnail images, and it must efficiently move the image data to the browser while maintaining the maximum flexibility in selecting images from the data store.

This recipe uses the same approach as in Recipe 15.3 where, with one page, an image is retrieved from the database and streamed to the browser, and a second page is used to generate the image requests and display the results, which is a set of thumbnails here. Additionally, the image retrieval page must scale the images to thumbnail size.

In our example that demonstrates the solution, the `Page_Load` method of the code-behind for the image building page (the `CH15ThumbnailFromDatabase` page) is modified to scale the full-size image retrieved from the database to the appropriate size for a thumbnail presentation.

The first step to scale the image is to create a`System.Drawing.Image` object from the byte array retrieved from the database. This requires loading the byte array into a`MemoryStream` and using the `FromStream` method of the `Image` class to create the image.

Next, we need to calculate how much to reduce the image. A constant is used to define the height of the thumbnail, and the width is calculated by determining how much the height is being reduced and multiplying the value times the width of the full-size image.

Reduce the height and width by the same scale factor to keep the aspect ratio correct. If the height or width is reduced by a different amount, the image will be distorted.

Handling mixed calculations of integers and doubles can result in unexpected results. Visual Basic is more tolerant and will allow the division of two integers with the quotient set to a variable of type double and result in the correct value. C# will return an integer result from the division of two integers, truncating the result. It is best to cast at least the numerator to the type of the resultant variable.

Now that the width and height of the thumbnail are determined, the`GetThumbnailImage` method of the full-size image can be used to return a scaled-down image to use as the thumbnail.

Once the thumbnail image is created, it can be loaded into a`MemoryStream` and written to the `Response` object in the same manner described in Recipe 15.3.

The web form used to display the thumbnails uses a `DataList` control to provide the ability to generate the list using data binding. The`DataList` is configured to display four columns horizontally by setting the `RepeatColumns` attribute to "4" and the`RepeatDirection` attribute to "`Horizontal`". This will start a new row in the table used to display the images after every fourth image.

A `HeaderTemplate` is used to label the table of images. The template can contain any HTML that can be properly displayed in a table. In this example, the template consists of a single table row containing a single cell with the heading for the table. The`colspan` attribute is set to "4" to cause the cell to span across all columns in the table, and the`align` attribute is set to "`center`" to center the heading.

An `ItemTemplate` is used to define an item displayed in the `DataList`. In this example, the template consists of an `img` tag that has the `ID` and `Runat` attributes set to provide access to the item from the code-behind.

The `Page_Load` method in the code-behind reads the list of images from the database and binds them to the `DataList` control. This is accomplished by opening a connection to the database, querying for a list of the image IDs, then setting the`DataSource` and calling the `dataBind` method of the `DataList`. The only data we need from the database is the list of IDs for the images. These will be used to set the image sources, and the reading of the image data will be done by the code described earlier.

The `src` attribute of the image tags used to display the thumbnail images is set in the `ItemDataBound` event of the DataList. The ItemDataBound event is executed for every item in the DataList, including the header. Therefore, it is important to check the item type since image tags only appear in the data items. Data items can be an Item or an AlternatingItem.

If the item is a data item, we need to get a reference to the image control in the item. This is accomplished by using the FindControl method of the item to locate the image control using the ID assigned in the *.aspx* file. This reference must be cast to an HtmlImage type since the return type of FindControl is an Object.

Once a reference to the image is obtained, the src attribute can be set to the name of the ASP.NET page that is used to generate the thumbnail image passing the ID of the image in the URL.

The ID of the image is obtained from the DataItem method of the item. This must be cast to a DbDataRecord type to allow "looking up" the ID of the image using the name of the column included in the SQL query used in the bindData method described earlier.

This example presents a useful method of displaying thumbnails of images stored in a database. When used with reasonably small images, the performance is acceptable for most applications. If the images are large, however, you may want to create the thumbnail images offline and store them in the database to avoid the performance hit for real-time conversion.

See Also

Recipes 15.1 and 15.3

Example 15-16. Page_Load method for generating thumbnail image (.vb)

```
Imports System.Configuration
Imports System.Data
Imports System.Data.Common
Imports System.Data.OleDb
Imports System.Drawing
Imports System.Drawing.Imaging
Imports System.IO

 …

  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   'height of the thumbnail created from the original image
   Const THUMBNAIL_HEIGHT As Integer = 75

   Dim dbConn As OleDbConnection
   Dim dc As OleDbCommand
   Dim imageData( ) As Byte
   Dim strConnection As String
   Dim strSQL As String
   Dim ms As MemoryStream
   Dim fullsizeImage As Image
   Dim thumbnailImage As Image
   Dim thumbnailWidth As Integer
   Dim imageID As String

   If (Not Page.IsPostBack) Then
    Try
    'get the ID of the image to retrieve from the database
    imageID = Request.QueryString(QS_IMAGE_ID)

    'get the connection string from web.config and open a connection
    'to the database
    strConnection = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDb.OleDbConnection(strConnection)
    dbConn.Open( )

    'build the query string and get the data from the database
    strSQL = "SELECT ImageData " &_
     "FROM BookImage " &_
     "WHERE BookImageID=" & imageID
    dc = New OleDbCommand(strSQL, dbConn)
    imageData = CType(dc.ExecuteScalar( ), Byte( ))

    'create an image from the byte array
    ms = New MemoryStream(imageData)
    fullsizeImage = System.Drawing.Image.FromStream(ms)

    'calculate the amount to shink the height and width
    thumbnailWidth = _
    CInt(Math.Round((CDbl(THUMBNAIL_HEIGHT) / _
          fullsizeImage.Height) * fullsizeImage.Width))

    'create the thumbnail image
    thumbnailImage = fullsizeImage.GetThumbnailImage(thumbnailWidth, _
                THUMBNAIL_HEIGHT, _
               Nothing, _
               IntPtr.Zero)
    'write thumbnail to the response object
    ms = New MemoryStream
    thumbnailImage.Save(ms, ImageFormat.Jpeg)
    Response.ContentType = "image/jpg"
    Response.BinaryWrite(ms.ToArray( ))

   Finally
    'clean up
    If (Not IsNothing(dbConn)) Then
    dbConn.Close( )
   End If
  End Try
 End If
End Sub 'Page_Load
```

Example 15-17. Page_Load method for generating thumbnail image (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;

 …

  protected void Page_Load(object sender, EventArgs e)
  {
  // height of the thumbnail created from the original image
  const int THUMBNAIL_HEIGHT = 75;

  OleDbConnection dbConn = null;
  OleDbCommand dc = null;
  byte[] imageData = null;
  String strConnection = null;
  String strSQL = null;
  MemoryStream ms = null;
  System.Drawing.Image fullsizeImage = null;
  System.Drawing.Image thumbnailImage = null;
  int thumbnailWidth;
  String imageID = null;

  if (!Page.IsPostBack)
  {
   try
   {
    // get the ID of the image to retrieve from the database
    imageID = Request.QueryString[QS_IMAGE_ID];

    // get the connection string from web.config and open a connection
    // to the database
    strConnection = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(strConnection);
    dbConn.Open( );

    // build the query string and get the data from the database
    strSQL = "SELECT ImageData " +
     "FROM BookImage " +
     "WHERE BookImageID=" + imageID;
    dc = new OleDbCommand(strSQL, dbConn);
```

```
     imageData = (byte[])(dc.ExecuteScalar( ));

     // create an image from the byte array
     ms = new MemoryStream(imageData);
     fullsizeImage = System.Drawing.Image.FromStream(ms);

     // calculate the amount to shink the height and width
     thumbnailWidth =
     Convert.ToInt32(Math.Round((Convert.ToDouble(THUMBNAIL_HEIGHT) /
       fullsizeImage.Height) * fullsizeImage.Width));

     // create the thumbnail image
     thumbnailImage = fullsizeImage.GetThumbnailImage(thumbnailWidth,
               THUMBNAIL_HEIGHT,
               null,
               IntPtr.Zero);

     // write thumbnail to the response object
     ms = new MemoryStream( );
     thumbnailImage.Save(ms, ImageFormat.Jpeg);
     Response.ContentType = "image/jpg";
     Response.BinaryWrite(ms.ToArray( ));
    }

    finally
    {
     if (dbConn != null)
     {
     dbConn.Close( );
    }
   }
  }
} // Page_Load
```

Example 15-18. Displaying thumbnail images (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH15TestThumbnailsFromDatabaseVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH15TestThumbnailsFromDatabaseVB"
 Title="Display Thumbnails" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Display Thumbnail Images (VB)
 </div>
 <div align="center">
  <asp:DataList ID="dlImages" Runat="server"
    RepeatColumns="4" RepeatDirection="Horizontal"
    RepeatLayout="Table" Width="50%"
    OnItemDataBound="dlImages_ItemDataBound">
   <HeaderTemplate>
    <tr>
    <td colspan="4" class="subHeading" align="center">
    Thumbnails of Images In Database<br /><br />
    </td>
    </tr>
   </HeaderTemplate>
   <ItemTemplate>
    <img id="imgThumbnail" runat="server" border="0" src="" alt="image"/>
   </ItemTemplate>
  </asp:DataList><
 </div>
</asp:Content>
```

Example 15-19. Displaying thumbnail images code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System.Configuration.ConfigurationManager
Imports System.Data
Imports System.Data.Common
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH15TestThumbnailsFromDatabaseVB.aspx
 ''' </summary>
 Partial Class CH15TestThumbnailsFromDatabaseVB
  Inherits System.Web.UI.Page
   '''*************************************************************************
```

```
''' <summary>
''' This routine provides the event handler for the page load event. It
''' is responsible for initializing the controls on the page.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Private Sub Page_Load(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles Me.Load
 Dim dbConn As OleDbConnection = Nothing
 Dim dc As OleDbCommand
 Dim dr As OleDbDataReader
 Dim strConnection As String
 Dim strSQL As String

 If (Not Page.IsPostBack) Then
  Try
  'get the connection string from web.config and open a connection
  'to the database
  strConnection = _
  ConnectionStrings("dbConnectionString").ConnectionString
  dbConn = New OleDb.OleDbConnection(strConnection)
  dbConn.Open( )

  'build the query string and get the data from the database
  strSQL = "SELECT BookImageID " &_
   "FROM BookImage"
  dc = New OleDbCommand(strSQL, dbConn)
  dr = dc.ExecuteReader( )

  'set the source of the data for the repeater control and bind it
  dlImages.DataSource = dr
  dlImages.DataBind( )

  Finally
  'clean up
  If (Not IsNothing(dbConn)) Then
  dbConn.Close( )
  End If
  End Try
 End If
End Sub 'Page_Load

'''*************************************************************************
''' <summary>
''' This routine is the event handler that is called for each item in the
''' datalist after a data bind occurs. It is responsible for setting the
''' source of the image tag to the URL of the page that will generate the
''' thumbnail images with the ID of the appropriate image for the item.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
```

```
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub dlImages_ItemDataBound(ByVal sender As Object, _
     ByVal e As System.Web.UI.WebControls.DataListItemEventArgs)

   Dim img As System.Web.UI.HtmlControls.HtmlImage

   'make sure this is an item in the data list (not header etc.)
   If ((e.Item.ItemType = ListItemType.Item) Or _
    (e.Item.ItemType = ListItemType.AlternatingItem)) Then
    'get a reference to the image used for the bar in the row
    img = CType(e.Item.FindControl("imgThumbnail"), _
    System.Web.UI.HtmlControls.HtmlImage)
    'set the source to the page that generates the thumbnail image
    img.Src = "CH15ThumbnailFromDatabaseVB.aspx?ImageID=" &_
      CStr(CType(e.Item.DataItem, DbDataRecord)("BookImageID"))
   End If
  End Sub 'dlImages_ItemDataBound
 End Class 'CH15TestThumbnailsFromDatabaseVB
End Namespace
```

Example 15-20. Displaying thumbnail images code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.Common;
using System.Data.OleDb;
using System.Web.UI.WebControls;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH15TestThumbnailsFromDatabaseCS.aspx
 /// </summary>
 public partial class CH15TestThumbnailsFromDatabaseCS : System.Web.UI.Page
 {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
```

```
 OleDbConnection dbConn = null;
 OleDbCommand dc = null;
 OleDbDataReader dr = null;
 String strConnection = null;
 String strSQL = null;

 if (!Page.IsPostBack)
 {
  try
  {
  // get the connection string from web.config and open a connection
  // to the database
  strConnection = ConfigurationManager.
  ConnectionStrings["dbConnectionString"].ConnectionString;
  dbConn = new OleDbConnection(strConnection);
  dbConn.Open( );
  // build the query string and get the data from the database
  strSQL = "SELECT BookImageID " +
   "FROM BookImage";
  dc = new OleDbCommand(strSQL, dbConn);
  dr = dc.ExecuteReader( );

  // set the source of the data for the repeater control and bind it
  dlImages.DataSource = dr;
  dlImages.DataBind( );
  }

  finally
  {
  // clean up
  if (dbConn != null)
  {
  dbConn.Close( );
  }
  }
 }
} // Page_Load

///***************************************************************************
/// <summary>
/// This routine is the event handler that is called for each item in the
/// datalist after a data bind occurs. It is responsible for setting the
/// source of the image tag to the URL of the page that will generate the
/// thumbnail images with the ID of the appropriate image for the item.
/// </summary>
///
/// <param name="sender">Set to the sender of the event</param>
/// <param name="e">Set to the event arguments</param>
protected void dlImages_ItemDataBound(Object sender,
  System.Web.UI.WebControls.DataListItemEventArgs e)
{
 System.Web.UI.HtmlControls.HtmlImage img = null;
```

```
    // make sure this is an item in the data list (not header etc.)
    if ((e.Item.ItemType == ListItemType.Item) ||
     (e.Item.ItemType == ListItemType.AlternatingItem))
    {
     // get a reference to the image used for the bar in the row
     img = (System.Web.UI.HtmlControls.HtmlImage)
       (e.Item.FindControl("imgThumbnail"));

     // set the source to the page that generates the thumbnail image
     img.Src = "CH15ThumbnailFromDatabaseCS.aspx?ImageID=" +
     (((DbDataRecord)(e.Item.DataItem))["BookImageID"]).ToString( );
    }
   } // dlImages_ItemDataBound
  } //  CH15TestThumbnailsFromDatabaseCS
}
```

PREV

PREV

Chapter 16. Caching

PREV

16.0 Introduction

ASP.NET gives you the ability to cache the output of pages or portions of pages in memory to improve performance. The main reason to cache is to reduce the latency and increase the scalability of an application while reducing the server resources required to deliver its page content. *Latency* is a measure of the time it takes for an application to respond to a user request. *Scalability* is the ability of an application to handle increased numbers of users. If a page is cached on the server, the rendered HTML stored in memory is served instead of a freshly generated page from the server. Because it takes less time for the client to get the page and display it, your web site will seem more responsive.

If pages are completely static, deciding to cache them is a no-brainer. But the decision gets trickier if pages must vary their content in response to one of the following:

- Query string parameters
- Client browser type (e.g., Internet Explorer, Netscape, and so on)
- Custom parameters
- Database content

Sometimes it makes sense to cache such pages, and sometimes not, as you will soon see.

ASP.NET provides the ability to cache data items to enhance performance. But what if the data items occasionally change? Does it still make sense to cache them? The answer is a definite yes, provided you know how to refresh the cache when the data changes.

ASP.NET 2.0 now allows for many new caching possibilities, including database-triggered cache invalidation and the ability to create custom cache dependencies. The recipes in this chapter cover how to use these new capabilities in the context of some of the more useful caching scenarios. Nevertheless, ASP.NET 2.0 caching is a large topic that will undoubtedly spawn many articles and to cover it exhaustively would be beyond the scope of this book.

Recipe 16.2. Caching Pages

Problem

You want to cache the pages in your application.

Solution

Add the `@ OutputCache` directive to the top of the *.aspx* file of each page you want to cache with the `VaryByParam` attribute set to "`None`":

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
        AutoEventWireup="false"
   CodeFile="CH16CachePageVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH16CachePageVB"
  Title="Caching ASPX Pages" %>
   <%@ OutputCache Duration="5" VaryByParam="None" %>

          …
```

Discussion

This recipe shows the minimum changes required to an *.aspx* file to cache a page of your ASP.NET application. Only one `@ OutputCache` directive can be included per page, and the `Duration` and `VaryByParam` attributes are required.

You specify how long the page is to be retained in the cache by setting the `Duration` attribute to the desired time in seconds. In our example directive, the page will be rendered on the first request for the page and placed in the cache. For five seconds, all subsequent requests for the page will be delivered from the cached copy. After five seconds, the page will again be rendered.

The duration can be set to any positive integer value (12,147,483,647), which allows caching a page for roughly 68 years. You may be tempted to use very large numbers; however, every cached page uses server resources, and, if the page is not needed frequently, you will tie up server resources unnecessarily.

The `VaryByParam` attribute is used to define parameters that determine which cached copy of a page should be sent to the client. If your page does not vary, set the `VaryByParam` attribute to "`None`" and a single copy of the page will be saved in the cache. For an example of caching multiple copies of a rendered page as a function of the values in the query string, see Recipe 16.2.

Not all pages should be cached. Caching pages used for data input or login functions can result in some odd behavior. Table 16-1 provides some guidelines for deciding when to cache a page.

Table 16-1. Suggestions for caching pages

Page type	Should it be cached?	Comment
Completely static	Yes	A single copy of the rendered page will be saved in the cache.
Contents change as a function of query string values	Possibly	Multiple copies of the rendered page will need to be saved in the cache (see Recipe 16.2).
Static, but rendered page varies by client browser	Yes	One copy of the rendered page will be saved in the cache for each major version of a browser (see Recipe 16.3).
Input form	No	Caching an input form page can result in the data entered by one user being displayed to another.
Dynamically created from database	Possibly	ASP.NET 2.0 provides the ability to have the cached data expire as a function of changes to the contents of the database (see Recipe 16.5).

See Also

Recipes 16.2, 16.3, and 16.5

PREV

Recipe 16.3. Caching Pages Based on Query String Parameter Values

Problem

You want to use page caching to improve the performance of your application, but the contents of your pages vary depending on the values of parameters in the query string.

Solution

Add the `@ OutputCache` directive to the *.aspx* file of each page you want to cache with the `VaryByParam` attribute set to the names of the parameters used in the query string:

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
        AutoEventWireup="false"
    CodeFile="CH16CachePageByQuerystringVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH16CachePageByQuerystringVB"
   Title="Cache Page By Querystring" %>
    <%@ OutputCache Duration="10" VaryByParam="DistrictID;SchoolID" %>

                …
```

Discussion

Programmers commonly pass information in the query string that is used to define what is displayed on a page. Because the page content is dependent on one or more parameters, the basic caching example shown in Recipe 16.1 cannot be used. Fortunately, ASP.NET provides the ability to cache multiple copies of a page by defining the dependent parameters.

You define the parameters ASP.NET will use to cache copies of a page by setting the`VaryByParam` attribute to a semicolon-separated list of the parameters used in the query string to define the page contents. For each page request, ASP.NET checks the values of the indicated parameters in the query string. If the parameter values match the parameter values of a copy of the page in the cache, the copy from the cache will be sent to the browser. If the parameter values do not match, the page will be rendered by your code, added to the cache, and sent to the browser.

In our example, the `VaryByParam` attribute is set to `DistrictID;SchoolID`, which causes ASP.NET to check the URL of each page request for the `DistrictID` and `SchoolID` parameters in the query string. An example URL is shown here:

```
http://michaelk3/aspnetcookbook2vb/CH16CachePageByQuerystringVB.aspx?
DistrictID=1&SchoolID=2
```

When ASP.NET receives the request for this page, it will check to see if a copy of the page exists in the cache for `DistrictID=1` and `SchoolID=2`. If the copy exists in the cache, it will be sent to the browser without rerendering the page, which can improve your application's responsiveness, especially if database access was required to render the page. If a copy for`DistrictID=1` and `SchoolID=2` is not in the cache, the page will be rendered by the server.

The `VaryByParam` attribute can be set to * to cause ASP.NET to cache a copy of the rendered page for every variation in parameters. This can result in caching more copies than you anticipate and generally should not be used.

Care should be taken in defining the duration for storing the page in the cache. If the duration is set to a large value and a large number of unique page requests are received within the duration period, server resources could be depleted.

This recipe deals with how to cache different versions of a rendered page as a function of its query string parameters. This same approach can be used to cache different versions of a page depending on form data when the page is posted back to the server. To cache versions as a function of posted parameters, use the names of the controls on the form (text boxes, checkboxes, and the like) in the `VaryByParam` attribute.

See Also

Recipe 16.1

PREV

Recipe 16.4. Caching Pages Based on Browser Type and Version

Problem

You have a page with static content that you want to cache, but the page is rendered differently for some browsers.

Solution

Add the `@ OutputCache` directive to the *.aspx* file of each page you want to cache with the `VaryByCustom` attribute set to "`browser`":

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
        AutoEventWireup="false"
    CodeFile="CH16CacheByCustomStringVB.aspx.vb"
     Inherits="ASPNetCookbook.VBExamples.CH16CacheByCustomStringVB"
   Title="Cache By Custom String" %>
    <%@ OutputCache Duration="10" VaryByParam="none"
                       VaryByCustom="browser" %>
           …
```

Discussion

The way in which a page is rendered often depends on the characteristics and version number of the browser making the request. ASP.NET handles most of this variation behind the scenes by sensing the browser type and its version and rendering the HTML and JavaScript in a suitable format.

The variation in a rendered page from one browser type to another can cause problems if you implement caching as described in Recipe 16.1. For example, if a request is made for a page of this type by Internet Explorer 6.x, the page is rendered and stored in the cache. If a request for the same page is made by a Netscape 4.x browser before the cached copy expires, the Internet Explorer 6.x version will be sent to the browser. This generally results in a poorly displayed or improperly functioning page.

ASP.NET provides the ability to cache browser-specific versions of the rendered page by setting the `VaryByCustom` attribute of the `OutputCache` element to "`browser`". When `VaryByCustom` is set to "`browser`", ASP.NET will check the browser name and major version of the browser making the page request (e.g., 4.x, 5.x, 6.x, etc.) to see if a copy of the rendered page is stored in the cache that matches them. If so, the cached version will be sent as the response. Otherwise, the page will be rendered as usual, stored in the cache, and delivered to the browser.

The only `VaryByCustom` attribute value that ASP.NET provides built-in support for is "`browser`"; however, with additional coding, this attribute can be used to control the caching of a page by any variation you choose. Recipe 16.4 provides an example of using the `VaryByCustom` attribute to cache a page based on the browser type and full version number.

> The `VaryByParam` attribute is required even though it is not being used in this example. Setting the value to "`none`" disables the caching by parameter. If you fail to include the `VaryByParam` attribute, a parse error will result when the page is requested.

See Also

Recipes 16.1 and 16.4

PREV

PREV

Recipe 16.5. Caching Pages Based on Developer-Defined Custom Strings

Problem

You have pages in your application you want to cache but ASP.NET does not provide built-in support for the dependencies you need, such as browser type and full version number.

Solution

Add the `@ OutputCache` directive at the top of the *.aspx* file of each page you want to cache. Set the `VaryByCustom` attribute to the name of a custom string, such as "`BrowserFullVersion`":

```
 <%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
         AutoEventWireup="false"
    CodeFile="CH16CacheByCustomStringVB.aspx.vb"
     Inherits="ASPNetCookbook.VBExamples.CH16CacheByCustomStringVB"
   Title="Cache By Custom String" %>
    <%@ OutputCache Duration="10" VaryByParam="none"
                       VaryByCustom="BrowserFullVersion" %>

            …
```

Override the `GetVaryByCustomString` method in *global.asax* and write code that builds a unique string for the value you have assigned to the `VaryByCustom` attribute. Examples 16-1 and 16-2 show VB and C# `GetVaryByCustomString` method to return a full browser version number.

Discussion

ASP.NET provides the ability to control the caching of pages as a function ofcustom strings that you provide, which gives you the ability to control the caching by variations not directly supported by ASP.NET. In the example we use to illustrate this solution, we show how to cache pages based on the browser type, its major version number (integer portion of version number), and its minor version number (the decimal portion of the version number).

The first step in this recipe is to add the `@ OutputCache` directive shown earlier to the *.aspx* file of the page you plan to cache. Set the `VaryByCustom` attribute to the name of the string that is to be used to determine caching. In our example, we have named the string "`BrowserFullVersion`".

Next, you need to override the GetVaryByCustomString method in *global.asax* to return the full browser version when the passed parameter (arg) is set to "BrowserFullVersion". This provides a unique string for each browser and version to allow ASP.NET to differentiate between the browsers and use the cached version of the page accordingly.

This technique is not limited to caching by browser version. You can use almost any information to identify pages that should be cached separately. For example, you could use a value stored in a cookie to determine the uniqueness of a page. The cookie collection is accessed through the Request object in the same manner as the Browser data (context.Request.Cookies).

One thing you cannot use is Session information. The reason for this is that when GetVaryByCustomString is called, session information has not been retrieved from session storage. The session ID is available but not the session data. If you need to use a value related to a specific session, the data will have to be stored in a cookie first and used as described previously.

Unlike other attributes in the @ OutputCache directive, the VaryByCustom attribute can contain only one value. It cannot be set to a semicolon-delimited string, because the entire unparsed value is passed to the GetVaryByCustomString method. Your code in the GetVaryByCustomString will have to perform the parsing if you want to use multiple values for a single page.

See Also

Search for "Caching Versions of a Page, Based on Custom Strings" in the MSDN library.

Example 16-1. GetVaryByCustomString method in global.asax (.vb)

```
'''***************************************************************************
''' <summary>
''' This routine provides the ability to set custom string values to
''' control the page or page fragment caching based on values assigned
''' to the VaryByCustom attribute of the OutputCache directive.
''' </summary>
'''
''' <param name="context">Set to the current Http context</param>
''' <param name="custom">
''' Set to the custom string that specifies which cached response is
''' used to respond to the current request
''' </param>
'''
''' <returns>
''' Set to a string containing the full browser verion
''' </returns>
Public Overrides Function GetVaryByCustomString( _
                            ByVal context As System.Web.HttpContext, _
```

```
    ByVal custom As String) As String
  Dim value As String = Nothing

  'if argument is requesting the full browser version, build a string
  'containing the browser name, major version, and minor version
  If (custom.Equals("BrowserFullVersion")) Then
    value = "BrowserFullVersion =" & _
            context.Request.Browser.Browser & _
   context.Request.Browser.MajorVersion.ToString() & "." & _
            context.Request.Browser.MinorVersion.ToString()
  End If

    Return (value)
End Function 'GetVaryByCustomString
```

Example 16-2. GetVaryByCustomString method in global.asax (.cs)

```
'''***********************************************************************
/// <summary>
/// This routine provides the ability to set custom string values to
/// control the page or page fragment caching based on values assigned
/// to the VaryByCustom attribute of the OutputCache directive.
/// </summary>
///
/// <param name="context">Set to the current Http context</param>
/// <param name="custom">
/// Set to the custom string that specifies which cached response is
/// used to respond to the current request
/// </param>
///
/// <returns>
/// Set to a string containing the full browser verion
/// </returns>
public override string GetVaryByCustomString(System.Web.HttpContext context,
                                             string custom)
{
  String value = null;

  // if argument is requesting the full browser version, build a string
  // containing the browser name, major version, and minor version
  if custom.Equals("BrowserFullVersion")
  {
    value = "BrowserFullVersion =" +
 context.Request.Browser.Browser +
 context.Request.Browser.MajorVersion.ToString() + "." +
 context.Request.Browser.MinorVersion.ToString();
  }
  return (value);
} // GetVaryByCustomString
```

PREV

PREV

Recipe 16.6. Caching Pages Based on Database Dependencies

Problem

You have pages in your application you want to cache but the data on the pages is retrieved from a database.

Solution

Configure SQL Server to support notifications when data changes (not required for SQL Server 2005 and SQL Server Express Edition), configure your application to use SQL dependencies by adding the `<sqlCacheDependency>` element to *web.config*, and add the `@ OutputCache` directive at the top of the *.aspx* file of each page you want to cache.

Discussion

ASP.NET 1.x provided many ways to cache data with dependencies, but caching data with database dependencies was not one of them. Fortunately, ASP.NET 2.0 has added the ability to cache pages with dependencies on data in a SQL Server database. SQL Server 7.0, 2000, 2005, MSDE, and SQL Server Express Edition are all supported, in one form or another.

The key to determining when cache content should be made to expire is knowing when the data in the database changes, which differs depending on the vintage of the SQL Server product your application uses.

SQL Server 2005 and SQL Server Express Edition provide notification events that can be used to notify an application the data has changed. This approach is referred to as*notification-based invalidation*. What's more, neither one requires any configuration changes to support this capability, so you can skip to the portion of the discussion where we describe adding the`<sqlCacheDependency>` element to *web.config*.

SQL Server 7, 2000, and MSDE do not support notification events. Instead, the ability to determine when data has changed is implemented by ASP.NET polling a table added to your database. This is referred to as *polling-based invalidation*. The table (`AspNet_SqlCacheTablesForChangeNotification`) contains a single row per table being monitored. When the data in a monitored table changes, a trigger is fired that changes the value for the `changeId` column in the row for the monitored table. The change in the `changeId` value will be what ASP.NET uses to determine if the data for the monitored table has changed.

If your application uses one of these earlier SQL Server products, yourdatabase will need to be altered to support notification when data changes.

Microsoft provides two options for configuring your database to support notifications. The first option is to use the `aspnet_regsql` command-line tool. You must first configure the database to support notifications and then specify the tables to be monitored. Use these commands to configure your database.

```
Run this command to configure your database for notifications:
     aspnet_regsql -S [server] -E -d [database] -ed

Run this command to configure a table in your database for notifications:
     aspnet_regsql -S [server] -E -d [database] -et -t [table]

   Where:
     [server] is the name or IP address of your SQL Server
  [database] is the name of the database to enable
  [table] is the name of the table to enable
```

To see all of the options available for configuring your database with the commandline tool, enter the following command:

```
aspnet_regsql -?
```

The second option for configuring your database for notifications is to use the methods of the `SqlCacheDependencyAdmin` class. Examples 16-3 (VB) and 16-4 (C#) show the code required to configure your database programmatically. Methods are shown for enabling the database, enabling tables, disabling tables, and disabling the database.

A reference will need to be added to your project for the `System.Web.Caching` and `System.Web.Data.SqlClient` assemblies to configure your database programmatically.

After configuring your database, you need to add the `<sqlCacheDependency>` element to your *web.config* to configure ASP.NET to poll your database:

```
<?xml  version="1.0"?>
<configuration>
        <system.web>

             …

    <caching>
                  <sqlCacheDependency enabled = "true" pollTime = "60000" >
```

```
                          <databases>
                               <add name="ASPNetCookbook"
                                       connectionStringName = "sqlConnectionString" />
                          </databases>
                     </sqlCacheDependency>
                </caching>

             …

           </system.web>
      </configuration>
```

The pollTime attribute defines the rate at which the AspNet_SqlCacheTablesForChangeNotification table is queried to determine if any data in the monitored tables has changed. The units are in milliseconds.

The name attribute of the <add> element of the <databases> element defines the name for the database that will be used in your application when defining SQL dependencies forpages. This does not have to be the database name.

The connectionStringName attribute must be set to the name of a connection string in the <connectionStrings> element of *web.config*.

A pollTime attribute can be added to the <add> element for a database to use different poll rates for each database. When the pollTime is specified for an individual database, it overrides the setting in the<sqlCacheDependency> element.

Now that everything is configured, you can cache pages using the database dependency by adding the @ OutputCache directive to your pages:

```
 <%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
         AutoEventWireup="false"
    CodeFile="CH16CachedByDatabaseDependencyVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH16CachedByDatabaseDependencyVB"
   Title="Cache By Database Dependency" %>
 <%@ OutputCache Duration="86400" VaryByParam="None"
                      SqlDependency="ASPNetCookbook:Book" %>

           …
```

The SqlDependency attribute is set to the name of the database (as defined in *web.config*) and the table that should be used as the dependency for the page. You can specify multiple databases and tables by providing a semicolon-separated list of databases and tables.

When using SQL Server 7, 2000, or MSDE, the finest granularity for determining if data has changed is the table. If any data is added, edited, or deleted in the table, ASP.NET will see it as a change and cause the cache to expire even if it is data that is not explicitly used in your page. This will even occur if a row is updated and the data is not changed.

The ability to cache data dependent on the database contents is a powerful addition to ASP.NET 2.0. This recipe only touches on the functionality available. Refer to Recipe 16.8 and the MSDN Library (search for `SqlCacheDependency`) for more information of the possibilities available in your application

See Also

Recipe16.8 and the MSDN Library (search for `SqlCacheDependency`)

Example 16-3. Methods for programmatically configuring your database for cache expiration (.vb)

```
'''*********************************************************************
''' <summary>
''' This routine provides the ability to enable the database defined in
''' the passed connection string for notifications that are used for
''' SQL dependency cache invalidations
''' </summary>
'''
''' <param name="connectionStr">
''' Set to the connection string to the database for which notifications
''' are to be enabled
''' </param>
Public Shared Sub enableDatabaseNotifications(ByVal connectionStr As String)
  Try
    'enable the database for notifications
 SqlCacheDependencyAdmin.EnableNotifications(connectionStr)

  Catch exc As UnauthorizedAccessException
    'user specified in the connection string does not have permissions
 'to create tables, etc. in the database
 'production application should handle this exception as required
 Throw

  Catch exc As SqlException
    'other SQL error occurred
 'production application should handle this exception as required
 Throw

  End Try
End Sub 'enableDatabaseNotifications

'''*************************************************************************
''' <summary>
''' This routine provides the ability to enable the the passed table
''' for notifications used for SQL dependency cache invalidations
''' </summary>
'''
''' <param name="connectionStr">
''' Set to the connection string to the database containing the table
''' to enable
''' </param>
''' <param name="tableName">
''' Set to the name of the table to be enabled for notifications
''' </param>
Public Shared Sub enableTableNotifications(ByVal connectionStr As String,_
          ByVal tableName As String)
  Try
    'enable the passed table for notifications
 SqlCacheDependencyAdmin.EnableTableForNotifications(connectionStr, _
                                                        tableName)
  Catch exc As UnauthorizedAccessException
    'user specified in the connection string does not have permissions
 'to create triggers, etc in the database
 'production application should handle this exception as required
 Throw

  Catch exc As SqlException
    'other SQL error occurred
 'production application should handle this exception as required
 Throw
  End Try
End Sub 'enableTableNotifications

'''*************************************************************************
''' <summary>
''' This routine provides the ability to disable notifications for the
''' database defined in the passed connection string
''' </summary>
'''
''' <param name="connectionStr">
''' Set to the connection string to the database for which notifications
''' are to be disabled
''' </param>
Public Shared Sub disableDatabaseNotifications(ByVal connectionStr As String)
  Try
    'disable the database notifications
 SqlCacheDependencyAdmin.DisableNotifications(connectionStr)

  Catch exc As UnauthorizedAccessException
    'user specified in the connection string does not have permissions
 'to delete tables, etc. in the database
 'production application should handle this exception as required
 Throw

  Catch exc As SqlException
    'other SQL error occurred
 'production application should handle this exception as required
 Throw
  End Try
End Sub 'disableDatabaseNotifications

'''*************************************************************************
''' <summary>
''' This routine provides the ability to disable the the passed table
''' for notifications used for SQL dependency cache invalidations
''' </summary>
'''
''' <param name="connectionStr">
''' Set to the connection string to the database containing the table
''' to disable
''' </param>
''' <param name="tableName">
''' Set to the name of the table to be disabled
''' </param>
Public Shared Sub disableTableNotifications(ByVal connectionStr As String,_
                                            ByVal tableName As String)
  Try
    'enable the passed table for notifications
 SqlCacheDependencyAdmin.DisableTableForNotifications(connectionStr,_
                                                          tableName)

  Catch exc As UnauthorizedAccessException
    'user specified in the connection string does not have permissions
 'to delete triggers, etc in the database
 'production application should handle this exception as required
 Throw

  Catch exc As SqlException
    'other SQL error occurred
    'production application should handle this exception as required
 Throw
  End Try
End Sub 'disableTableNotifications
```

Example 16-4. Methods for programmatically configuring your database for cache expiration (.cs)

```
'''*************************************************************************
```

```
/// <summary>
/// This routine provides the ability to enable the database defined in
/// the passed connection string for notifications that are used for
/// SQL dependency cache invalidations
/// </summary>
///
/// <param name="connectionStr">
/// Set to the connection string to the database for which notifications
/// are to be enabled
/// </param>
public static void enableDatabaseNotifications(String connectionStr)
{
  try
  {
    // enable the database for notifications
 SqlCacheDependencyAdmin.EnableNotifications(connectionStr);
  }

  catch (UnauthorizedAccessException exc)
  {
    // user specified in the connection string does not have permissions
 // to create tables, etc. in the database
 // production application should handle this exception as required
 throw;
  }

  catch (SqlException exc)
  {
    // other SQL error occurred
 // production application should handle this exception as required
 throw;
  }
} // enableDatabaseNotifications

'''*************************************************************************
/// <summary>
/// This routine provides the ability to enable the the passed table
/// for notifications used for SQL dependency cache invalidations
/// </summary>
///
/// <param name="connectionStr">
/// Set to the connection string to the database containing the table
/// to enable
/// </param>
/// <param name="tableName">
/// Set to the name of the table to be enabled for notifications
/// </param>
public static void enableTableNotifications(String connectionStr,
                                           String tableName)
{
  try
  {
    // enable the passed table for notifications
 SqlCacheDependencyAdmin.EnableTableForNotifications(connectionStr,
                                                        tableName);
  }

  catch (UnauthorizedAccessException exc)
  {
    // user specified in the connection string does not have permissions
 // to create triggers, etc in the database
 // production application should handle this exception as required
 throw;
  }

  catch (SqlException exc)
  {
    // other SQL error occurred
 // production application should handle this exception as required
 throw;
  }
} // enableTableNotifications

'''**********************************************************************
/// <summary>
/// This routine provides the ability to disable notifications for the
/// database defined in the passed connection string
/// </summary>
///
/// <param name="connectionStr">
/// Set to the connection string to the database for which notifications
/// are to be disabled
/// </param>
public static void disableDatabaseNotifications(String connectionStr)
{
  try
  {
    // disable the database notifications
 SqlCacheDependencyAdmin.DisableNotifications(connectionStr);
  }

  catch (UnauthorizedAccessException exc)
  {
     // user specified in the connection string does not have permissions
         // to delete tables, etc. in the database
     // production application should handle this exception as required
  throw;
  }

  catch (SqlException exc)
  {
    // other SQL error occurred
 // production application should handle this exception as required
 throw;
```

```
  }
    } // disableDatabaseNotifications

'''**************************************************************************
/// <summary>
/// This routine provides the ability to disable the the passed table
/// for notifications used for SQL dependency cache invalidations
/// </summary>
///
/// <param name="connectionStr">
/// Set to the connection string to the database containing the table
/// to disable
/// </param>
/// <param name="tableName">
/// Set to the name of the table to be disabled
/// </param>
public static void disableTableNotifications(String connectionStr,
                                             String tableName)
{
  try
  {
    // enable the passed table for notifications
 SqlCacheDependencyAdmin.DisableTableForNotifications(connectionStr,
                                                         tableName);
  }

  catch (UnauthorizedAccessException exc)
  {
     // user specified in the connection string does not have permissions
  // to delete triggers, etc in the database
  // production application should handle this exception as required
  throw;
  }

  catch (SqlException exception)
  {
     // other SQL error occurred
  // production application should handle this exception as required
  throw;
  }
} // disableTableNotifications
```

PREV

Recipe 16.7. Caching User Controls

Problem

You have data entry pages in your application that cannot be cached, but the pages containuser controls that do not change and you want to cache them.

Solution

Add the `@ OutputCache` directive at the top of each *.ascx* file of the user controls you want to cache:

```
<%@ OutputCache Duration="5" VaryByParam="none" %>
```

Discussion

This solution is the same as that described in Recipe 16.1, except that it is applied to a user control. fact, all of the solutions described previously in this chapter can be applied to user controls.

User controls have one caching feature unavailable to pages: a user control can be cached as a function of its properties. This is quite handy because many times a user control, like a header or global navigation bar, varies as a function of its properties, not the parameters passed to the page or which it is used.

To see this in action for yourself, we suggest implementing the user control example described in Recipe 5.1 and adding the`@ OutputCache` directive shown here at the top of the *.ascx* file for the header user control:

```
<%@ OutputCache Duration="15"
                    VaryByControl="headerImage;dividerColor;dividerHeight" %>
```

A copy of the user control will be cached for every combination of the`headerImage, dividerColor`, and `dividerHeight` property values used in the application.

Though this example is not all that useful, since the header on a page is generally consistent within an application, the example demonstrates the ability to cache user controls as a function of the property values. A more practical but longer example would be the implementation of a global navigation bar that incorporates different images as a function of the page currently displayed in the application.

> When caching user controls in pages, verify that the control is valid before accessing any properties of the control. When the page is first displayed, the user control will be created and will be available to the page. Subsequent page requests, while the user control is residing in the cache, will not create the control, and the variable used to reference the control will be set to "`Nothing`" (VB) or "`null`" (C#). If your code attempts to access the user control without first checking the control's validity, a null reference exception will be thrown.

See Also

Recipe 5.1

PREV

Recipe 16.8. Caching Application Data

Problem

Your application draws on data that is expensive to create from a performance perspective, so you want to store it in memory to be accessed by users throughout the lifetime of the application. The problem is that the data changes occasionally and you need to refresh the data after it changes.

Solution

Place the data in the Cache object with a dependency set to the source of the data so the data will be reloaded when it changes. Examples 16-5 and 16-6 show the code we've written to demonstrate this solution. In this case, we have created a class, CH16CacheService, with methods that place some sample XML book data in the `Cache` object. In our example, the book data is automatically removed from the cache any time the XML file is changed.

Discussion

The `Cache` object in ASP.NET provides the ability to store application data in a manner similar to the storing of data in the `Application` object. The `Cache` object, unlike the `Application` object, lets you specify that the cached data is to be replaced at a specified time or whenever there is a change to the original source of the data.

Examples 16-5 (VB) and 16-6 (C#) show the CH16CacheService class with two methods we created for this example. The first method (`getBookData`) provides access to the data stored in the `Cache` object with the appropriate checking (to ensure the data is still valid) and reloading as required. The `getBookData` method performs the following operations:

1. Gets a reference to the book data in the `Cache` object
2. Checks the reference to ensure the data is still valid and reloads the data using the `loadBookDataInCache` method if it is not
3. Returns a reference to the book data

The second method (`loadBookDataInCache`) provides the ability to load the book data initially into the `Cache`. It performs the following operations:

1. Reads the book data from the XML file into a `DataSet`

2. Stores the DataTable in the DataSet in the Cache object

3. Returns a reference to the book data

When the DataTable in the DataSet is stored in the Cache object, three parameters are passed to the Insert method of the Cache object. The first parameter is the "key" value used to access the data in the cache. A constant is used here since the key value is needed in several places in the code. The second parameter is the DataTable containing the book data. The third parameter is the dependency on the XML file that was the original source of the data. By adding this dependency, the data is automatically removed from the cache any time the XML file is changed.

```
context.Cache.Insert(CAC_BOOK_DATA, _
                        ds.Tables(BOOK_TABLE), _
    New CacheDependency(xmlFilename))
```

```
context.Cache.Insert(CAC_BOOK_DATA,
                        ds.Tables[BOOK_TABLE],
    new CacheDependency(xmlFilename));
```

A DataTable is being stored in the Cache object instead of a DataSet because a DataTable uses less system resources and the extra functionality of a DataSet is not needed.

The getBookData method provides access to the data stored in the Cache object with the appropriate checking to ensure the data is still valid and reloading it as required.

The first step to retrieving the cached data is to get a reference to the book data in the Cache object:

```
bookData = CType(context.Cache.Item(CAC_BOOK_DATA), _
                    DataTable)
```

```
bookData = (DataTable)(context.Cache[CAC_BOOK_DATA]);
```

Next, the reference must be checked to ensure the data is still valid and, if it is not, the data must be reloaded using the loadBookDataInCache method:

```
If (IsNothing(bookData)) Then
```

```
     'data is not in the cache so load it
     bookData = loadBookDataInCache(context)
   End If
```

C#

```
 if (bookData == null)
 {
      // data is not in the cache so load it
   bookData = loadBookDataInCache(context);
 }
```

The caching object provides many additional features not described in our example, including the ability to replace the data based on a specified time and the ability to have one object in the cache b dependent on another object in the cache. For more information on these topics, refer to the MSDN documentation on the `Cache` and `CacheDependency` objects. For an example of caching application data dependent on data in a database, see Recipe 16.8.

Avoiding Race Conditions

The code shown in our example is designed to avoid a race condition that can result in a difficult-to-find error. The race condition is best described by example. Assume the following code was used (VB code shown):

```
1  If  (IsNothing(context.Cache.Item(CAC_BOOK_DATA))  then
2      loadBookDataInCache(context)
3   End If
4    bookData  =  Ctype(context.Cache.Item(CAC_BOOK_DATA),  _
5                        DataTable)
```

The code shown on line 1 checks to see if the book data exists in the cache, but it does not retrieve the data. If the data is valid, the next line of code to execute will be line 4. If the dependency causes the data to be removed from the cache between the execution of lines 1 and 4, a null reference exception will be thrown at line 4 because the data is no longer in the cache.

This example precludes the problem by retrieving the data as the first step and then checking to see if the data is valid. Because you have a copy of the data, you do not care if the data is removed from the cache. Likewise, the `loadBookDataInCache` method returns the data to avoid the same race condition problem.

See Also

Recipe 16.8 and MSDN documentation on theCache and CacheDependency

Example 16-5. Using application data in cache (.vb)

```
Option Explicit On
Option Strict On

Imports System.Data
Imports System.Data.SqlClient

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides caching functions used in chapter 16
  ''' </summary>
  Public Class CH16CacheService

    'the following constant used to define the name of the variable used to
 'store the book data in the cache object
 Private Const CAC_BOOK_DATA As String = "BookData"

 '''**********************************************************************
 ''' <summary>
 ''' This routine reads the book data from an XML file and places it in
 ''' the cache object.
 ''' </summary>
 '''
 ''' <param name="context">Set to the current Http context</param>
 '''
 ''' <returns>DataTable containing book data loaded into the cache</returns>
 Private Shared Function loadBookDataInCache(ByVal context As HttpContext) _
                  As DataTable
      Const BOOK_TABLE As String = "Book"

   Dim xmlFilename As String
   Dim ds As DataSet

   'read book data from XML file
   xmlFilename = context.Server.MapPath("xml") & "\books.xml"
   ds = New DataSet
   ds.ReadXml(xmlFilename)

   'store datatable with book data in cache scope with a
   'dependency to the original XML file
   context.Cache.Insert(CAC_BOOK_DATA, _
                        ds.Tables(BOOK_TABLE), _
       New CacheDependency(xmlFilename))
      'return the data added to the cache
   Return (ds.Tables(BOOK_TABLE))
    End Function 'loadBookDataInCache

 '''*********************************************************************
 ''' <summary>
 ''' This routine gets the book data from cache and reloads the cache if
 ''' required.
 ''' </summary>
 '''
 ''' <param name="context">Set to the current Http context</param>
 '''
 ''' <returns>DataTable containing book data from the cache</returns>
 Public Shared Function getBookData(ByVal context As HttpContext) _
           As DataTable
      Dim bookData As DataTable

   'get the book data from the cache
   bookData = CType(context.Cache.Item(CAC_BOOK_DATA), _
                       DataTable)

      'make sure the data is valid
   If (IsNothing(bookData)) Then
        'data is not in the cache so load it
  bookData = loadBookDataInCache(context)
      End If

   Return (bookData)
    End Function 'getBookData
  End Class 'CH16CacheService
End Namespace
```

Example 16-6. Using application data in cache (.cs)

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web;
using System.Web.Caching;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides caching functions used in chapter 16
  /// </summary>
  public class CH16CacheService
  {
    // the following constant used to define the name of the variable used to
 // store the book data in the cache object
 private const String CAC_BOOK_DATA = "BookData";

 '''*************************************************************************
 /// <summary>
 /// This routine reads the book data from an XML file and places it in
 /// the cache object.
 /// </summary>
 ///
 /// <param name="context">Set to the current Http context</param>
 ///
 /// <returns>DataTable containing book data loaded into the cache</returns>
 private static DataTable loadBookDataInCache(HttpContext context)
 {
      const String BOOK_TABLE = "Book";

   String xmlFilename = null;
   DataSet ds = null;

   // read book data from XML file
   xmlFilename = context.Server.MapPath("xml") + "\\books.xml";
   ds = new DataSet();
   ds.ReadXml(xmlFilename);

   // store datatable with book data in cache scope with a
   // dependency to the original XML file
   context.Cache.Insert(CAC_BOOK_DATA,
                           ds.Tables[BOOK_TABLE],
       new CacheDependency(xmlFilename));

      // return the data added to the cache
   return (ds.Tables[BOOK_TABLE]);
   } // loadBookDataInCache

   '''*************************************************************************
   /// <summary>
   /// This routine gets the book data from cache and reloads the cache if
   /// required.
   /// </summary>
   ///
   /// <param name="context">Set to the current Http context</param>
   ///
   /// <returns>DataTable containing book data from the cache</returns>
   public static DataTable getBookData(HttpContext context)
   {
     DataTable bookData = null;

  // get the book data from the cache
  bookData = (DataTable)(context.Cache[CAC_BOOK_DATA]);

  // make sure the data is valid
```

```
   if (bookData == null)
   {
        // data is not in the cache so load it
     bookData = loadBookDataInCache(context);
      }
   return (bookData);
   } // getBookData
 } // CH16CacheService
}
```

PREV

PREV

Recipe 16.9. Caching Application Data Based on Database Dependencies

Problem

Your application draws on data stored in database that is expensive to create from a performance perspective, so you want to store the data in memory, where it can be accessed by users throughout the lifetime of the application. The problem is that the data changes occasionally and you need to refresh the data when it changes.

Solution

Configure your SQL Server database and add the `<sqlCacheDependency>` element to *web.config* as described in Recipe 16.5, store the data in the Cache with a `SqlCache-Dependency`, and access the data in the Cache as required in your application.

The application we have implemented to demonstrate the solution is shown in Examples 16-7, 16-8 through 16-9. Example 16-7 shows the *.aspx* file used to display the cached data, and Examples 16-8 (VB) and 16-9 (C#) show the code-behind classes for the page.

Discussion

ASP.NET 2.0 has added a `SqlCacheDependency` class that can be used to create a dependency to data in a database. The `SqlCacheDependency` class uses the same infrastructure described in Recipe 16.5 tc determine if the data in the database has changed and to cause the data to expire in the Cache wher it has.

Once the configuration of SQL Server and the `<sqlCacheDependency>` element has been added to your *web.config*, as described in Recipe 16.5, adding data to the Cache with a dependency to data in the database is easy. All that is required is to create an instance of a `SqlCacheDependency` class defining the database (as delineated in *web.config*) and table for the dependency and pass this instance when inserting the data in the Cache:

```
sqlDep = New SqlCacheDependency("ASPNetCookbook", _
                                   "Book")

   Context.Cache.Insert(CAC_BOOK_DATA, _
                     bookData, _
    sqlDep)
   sqlDep = new SqlCacheDependency("ASPNetCookbook",
                                   "Book");

   Context.Cache.Insert(CAC_BOOK_DATA,
                        bookData,
    sqlDep);
```

If you are using SQL Server 2005, you can create a `SqlCacheDependency` object by passing a `SqlCommand` object to the constructor. Using this approach provides a more explicit definition of the data that is monitored and used for expiring the data in the Cache because a `SqlCommand` can contain any SQL statement complete with a `WHERE` clause:

```
 cmdText = "SELECT Title, PublishDate, ListPrice " & _
              "FROM Book " & _
      "WHERE PublishDate>='01/01/2005'"
    sqlCmd = New SqlCommand(cmdText, _
                            dbConn)
    sqlDep = New SqlCacheDependency(sqlCmd)
```

```
 cmdText = "SELECT Title, PublishDate, ListPrice " +
              "FROM Book " +
      "WHERE PublishDate>='01/01/2005'";
    sqlCmd = New SqlCommand(cmdText,
                            dbConn);
    sqlDep = New SqlCacheDependency(sqlCmd);
```

As this recipe shows, being able to cache application data based on a database dependency has become easy in ASP.NET 2.0. At the same time, it represents an improvement in the way that ASP.NET applications can be built and deployed.

The caching of data based on database dependencies should not be used for in-memory databases, since it results in replicating the data in memory with no appreciable performance gain.

See Also

Recipe 16.5

Example 16-7. Expiring cache data with a database dependency (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
     AutoEventWireup="false"
   CodeFile="CH16ExpireDatabaseDependentCacheDataVB.aspx.vb"
   Inherits="ASPNetCookbook.VBExamples.CH16ExpireDatabaseDependentCacheDataVB"
  Title="Cache Data With Database Dependency" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
     <div align="center" class="pageHeading">
          Cache Data With Database Dependency (VB)
     </div>
  <div align="center" class="labelText">
          <asp:Label ID="labCacheStatus" runat="server" />
     </div>
  <br />
  <asp:GridView ID="gvBooks" Runat="Server"
                      AllowPaging="false"
      AllowSorting="false"
      AutoGenerateColumns="false"
       BorderColor="#000080"
      BorderStyle="Solid"
      BorderWidth="2px"
      Caption=""
      HorizontalAlign="Center"
      Width="90%" >
          <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
    <RowStyle cssClass="tableCellNormal" />
    <AlternatingRowStyle cssClass="tableCellAlternating" />
    <Columns>
               <asp:BoundField DataField="Title"
                                  HeaderText="Title" />
               <asp:BoundField HeaderText="Publish Date"
                                  DataField="PublishDate"
       ItemStyle-HorizontalAlign="Center"
        DataFormatString="{0:MMM dd, yyyy}" />
               <asp:BoundField HeaderText="List Price"
                                  DataField="ListPrice"
       ItemStyle-HorizontalAlign="Center"
        DataFormatString="{0:C2}" />
          </Columns>
     </asp:GridView>
</asp:Content>
```

Example 16-8. Expiring cache data with a database dependency code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System.Configuration
Imports System.Data
Imports System.Data.SqlClient

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code-behind for
  ''' CH16ExpireDatabaseDependentCacheDataVB.aspx
  ''' </summary>
  Partial Class CH16ExpireDatabaseDependentCacheDataVB
    Inherits System.Web.UI.Page

    Private Const CAC_BOOK_DATA As String = "CH16BookData"

 '''*********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Private Sub Page_Load(ByVal sender As Object, _
                         ByVal e As System.EventArgs) Handles Me.Load
      Dim bookData As DataTable
   Dim dbConn As SqlConnection = Nothing
   Dim da As SqlDataAdapter = Nothing
   Dim strConnection As String
   Dim cmdText As String
   Dim sqlDep As SqlCacheDependency = Nothing
   Dim retrievedFromCache As Boolean = True

   If (Not Page.IsPostBack) Then
        'get the book data from the cache
  bookData = CType(Cache(CAC_BOOK_DATA), _
                       DataTable)

        'make sure the data is valid
  If (IsNothing(bookData)) Then
          'data is not in the cache so load it
    retrievedFromCache = False
    strConnection = ConfigurationManager. _
              ConnectionStrings("sqlConnectionString").ConnectionString
          dbConn = New SqlConnection(strConnection)
    dbConn.Open()

    cmdText = "SELECT Title, PublishDate, ListPrice " & _
              "FROM Book " & _
    "ORDER BY Title"
    da = New SqlDataAdapter(cmdText, dbConn)
    bookData = New DataTable
    da.Fill(bookData)

                'create the SQL dependency to the database table where the
    'data was retrieved
    sqlDep = New SqlCacheDependency("ASPNetCookbook", _
                                          "Book")

                'store book data in cache with a SQL dependency
    Context.Cache.Insert(CAC_BOOK_DATA, _
                                  bookData, _
       sqlDep)
         End If 'If (IsNothing(bookData))

    'set the source of the data for the gridview control and bind it
    gvBooks.DataSource = bookData
    gvBooks.DataBind()

    'set the label indicating where the data came from
    If (retrievedFromCache) Then
          labCacheStatus.Text = "Data was retrieved from the cache"
        Else
          labCacheStatus.Text = "Data was retrieved from the database"
        End If
      End If 'If (Not Page.IsPostBack)
    End Sub 'Page_Load
  End Class 'CH16ExpireDatabaseDependentCacheDataVB
End Namespace
```

Example 16-9. Expiring cache data with a database dependency code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Web.Caching;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code-behind for
  /// CH16ExpireDatabaseDependentCacheDataCS.aspx
  /// </summary>
  public partial class CH16ExpireDatabaseDependentCacheDataCS :
    System.Web.UI.Page
  {
     private const String CAC_BOOK_DATA = "CH16BookData";

 '''*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
      DataTable bookData = null;
   SqlConnection dbConn = null;
   SqlDataAdapter da = null;
   String strConnection;
   String cmdText;
   SqlCacheDependency sqlDep = null;
   Boolean retrievedFromCache = true;

   if (!Page.IsPostBack)
   {
        // get the book data from the cache
  bookData = (DataTable)(Cache[CAC_BOOK_DATA]);

  // make sure the data is valid
  if (bookData == null)
  {
          // data is not in the cache so load it
    retrievedFromCache = false;
    strConnection = ConfigurationManager.
              ConnectionStrings["sqlConnectionString"].ConnectionString;
          dbConn = new SqlConnection(strConnection);
    dbConn.Open();

    cmdText = "SELECT Title, PublishDate, ListPrice " +
                    "FROM Book " +
    "ORDER BY Title";

          da = new SqlDataAdapter(cmdText, dbConn);
    bookData = new DataTable();
    da.Fill(bookData);

    // create the SQL dependency to the database table where the
    // data was retrieved
    sqlDep = new SqlCacheDependency("ASPNetCookbook",
                                          "Book");

          // store book data in cache with a SQL dependency
    Context.Cache.Insert(CAC_BOOK_DATA,
                               bookData,
       sqlDep);
        } // If (IsNothing(bookData))

  // set the source of the data for the gridview control and bind it
  gvBooks.DataSource = bookData;
  gvBooks.DataBind();
        // set the label indicating where the data came from
  if (retrievedFromCache)
  {
          labCacheStatus.Text = "Data was retrieved from the cache";
        }
  else
  {
          labCacheStatus.Text = "Data was retrieved from the database";
        }
      } // If (Not Page.IsPostBack)
    } // Page_Load
  } // CH16ExpireDatabaseDependentCacheDataCS
}
```

Recipe 16.10. Caching Data Sources

Problem

You are using data sources in your application and you would like to cache the data source to improve th
performance of your application.

Solution

Set the `EnableCaching` property of the data source to `true` and set the `CacheDuration` property to the desired expiration time:

VB

```
dSource.EnableCaching = True
dSource.CacheDuration = 5
```

```
dSource.EnableCaching = true;
dSource.CacheDuration = 5;
```

Discussion

The data sources available in ASP.NET 2.0 (`XmlDataSource, ObjectDataSource` , and `SqlDataSource`) provide built-in support for caching. By using this built-in support, the data source will automatically cac
the data it pulls from the data store. In other words, you do not need to handle the insertion and retriev
of the data source in the Cache explicitly, which simplifies the code in your application.

To use the built-in caching support in data sources, you need to set a couple of properties. First, you ne
to set the `EnableCaching` property to `TRue` to enable the caching of the data source.

Next, you need to define when the cached data expires by setting the`CacheDuration` property to the number of seconds after which the cached data is invalidated.

Optionally, you can set the `CacheExpirationPolicy` property to alter the way the `CacheDuration` time is used to cause the cache data to expire. Setting the `CacheExpirationPolicy` to `DataSourceCacheExpiry.Absolute` causes the cached data source to expire the number of seconds define
in the `CacheDuration` after the data source is first created. Setting the `CacheExpirationPolicy` to `DataSourceCacheExpiry.Sliding` causes the time to reset each time the data source is accessed, creatinç
sliding expiration window.

If you are using a `SqlDataSource` or an `ObjectDataSource` , you can set the `SqlCacheDependency` property
the database and table on which the data source is dependent to cause the cached data to expire when

the data in the database change. If the data is dependent on more than one database and/or table, the dependencies can be specified in a semicolon-delimited list:

VB

```
'Use the following if the data is dependent on a single database and table
dSource.SqlCacheDependency = "ASPNetCookbook:Book"

'Use the following if the data is dependent on more than one database
'and/or table
dSource.SqlCacheDependency = "ASPNetCookbook:Book;ASPNetCookbook:Problem"
```

C#

```
// Use the following if the data is dependent on a single database and table
dSource.SqlCacheDependency = "ASPNetCookbook:Book";

// Use the following if the data is dependent on more than one database // and/or table
dSource.SqlCacheDependency = "ASPNetCookbook:Book;ASPNetCookbook:Problem";
```

> Setting the `SqlCacheDependency` requires configuring your SQL Server database and *web.config* , as described in Recipe 16.5.

See Also

Recipe 16.5

Chapter 17. Internationalization

17.0 Introduction

Internationalizing an application means making it world-ready. There are two processes involved:

Globalization

The process of designing an application so that it can adapt to different cultures

Localization

The process of translating the application's resources for a specific culture

Internationalizing an application requires explicit planning during the initial design. It generally consists of three steps:

1. Designing for globalization by ensuring the application is culture and language neutral. In other words, any content whose display depends on culture or language cannot be hardcoded in the HTML or program code.
2. Designing for localization by ensuring that no data that is culture-or language-specific is contained in the code and that the required data is obtained from resource files.
3. Creating specific resource files to support each culture and language.

A few samples of internationalization are included. If you need to internationalize an application, you would do well to obtain training or purchase one of the many books dedicated to the subject.

Recipe 17.2. Localizing Request/Response Encoding

Problem

You are developing an application for a specific region, and you want to tell the browser which character set to use in rendering the page.

Solution

Set the `requestEncoding` and `responseEncoding` attributes of the `<globalization>` element in *web.config* to the desired character set:

```
 <system.web>
       <globalization requestEncoding="iso-8859-1" responseEncoding="iso-8859-1" />
    </system.web>
```

Discussion

The HTTP header returned to the browser in response to a request contains information not displayed but, nevertheless, controls how the browser displays the content it receives. Included in the header is information that specifies which character set has been used to encode the response data and, by implication, which character set the browser should use to display it.

The request and response encoding information for a page request can be viewed by setting the `trace` attribute of the `@ Page` directive to `true` and viewing the Request Details section.

ASP.NET lets you specify the character set used to encode the response data using the `responseEncoding` attribute of the `<globalization>` element in the *web.config* file, as shown earlier. The `responseEncoding` attribute can be set to any valid character set. Table 17-1 lists some of the more common character sets used for European languages (English, French, German, and others).

Table 17-1. Common character sets

Character set name	Description
iso-8859-1	Commonly called Latin 1; covers the Western European languages
iso-8859-2	Commonly called Latin 2; covers the Central and Eastern European languages
Windows-1252	Windows version of the character set covering the Western European languages
utf-8	Technically not a character set but an encoding scheme to provide the ability to encode Unicode characters as a sequence of bytes

The character set information provided in the response headers is not guaranteed to be honored because the client machine may not have the suggested character set installed. If Internet Explorer is being used, it will prompt the user to install the required character set. If the character set is not installed, IE will use the character set defined by the computer's region locale setting. Other browsers may respond differently.

The `requestEncoding` attribute is used to specify the assumed encoding for incomingrequests. This includes posted data and data passed in the URL. Generally, the`requestEncoding` attribute is set to the same character set as the `responseEncoding` attribute.

If your *web.config* file does not contain a `<globalization>` element with `responseEncoding` and `requestEncoding` attributes set to a particular character set, the values defined in the *machine.config* file are used instead. By default, requests and responses are encoded in utf-8 format. If your *machine.config* file does not contain a `<globalization>` element with the `responseEncoding` and `requestEncoding` attributes set to the given character set, the encoding defaults to the computer's region locale setting.

The character set can be defined on a page-by-page basis by placing the`<meta>` tag shown here in the header section of the HTML (or *.aspx*) file:

```
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
```

This approach is not recommended, though, for two reasons. First, you need to add a `<meta>` tag to every page of your application, which implies some associated maintenance should any changes be required. Second, some browsers interpret `<meta>` tags only after they have rendered a page, which can cause the page to be rendered twice.

See Also

The "`<globalization>` Element" in the MSDN Library

PREV

Recipe 17.3. Providing Multiple Language Support

Problem

You want to support multiple languages in your application without developing multiple versions of each

Solution

Use resource files to provide the text for each user interface element in the languages you wish to supp attributes to the static controls in your *.aspx* files to set their text automatically from the resource files a runtime, set the `culture` and `uiCulture` attributes of the `<globalization>` element in *web.config* to `auto` set the values for dynamic controls in the code-behind class.

Use Visual Studio 2005 to create the root resource file and add the resource attributes to the controls in *.aspx* files as follows:

1. Open the *.aspx* file that you want to localize.
2. Switch to `Design` mode.
3. Select Tools Generate Local Resource from the menu. Visual Studio 2005 will add a `meta:resour` attribute to all server controls in the file and create a root resource file with entries for each of the controls.
4. Duplicate the root resource file for each language and culture to be supported by your application, the values to the resource items as appropriate for the language.
5. Set the `culture` and `uiCulture` attributes of the `<globalization>` element in *web.config* to `auto` :

```
<globalization culture="auto" uiCulture="auto" />
```

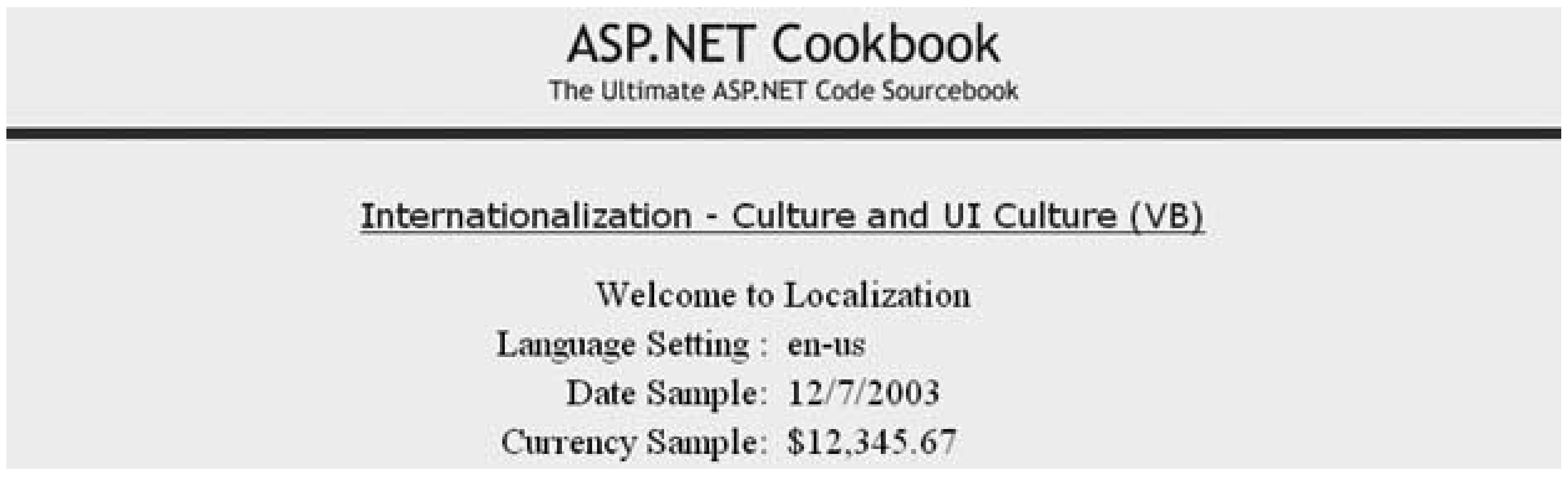
In the code-behind class for each page that needs to support multiplelanguages, use the .NET language your choice to:

1. Set the values of controls used for date and currency.
2. Set the text of any controls that need to be set programmatically.

Examples 17-1 , 17-2 through 17-3 show the *.aspx* file and VB and C# code-behind files for an applicatio written to demonstrate this solution. The output is shown in Figures 17-1(English) and 17-2 (German).

Examples 17-4 and 17-5 show the English and German resource files used in this example.

Figure 17-1. Multiple language outputEnglish

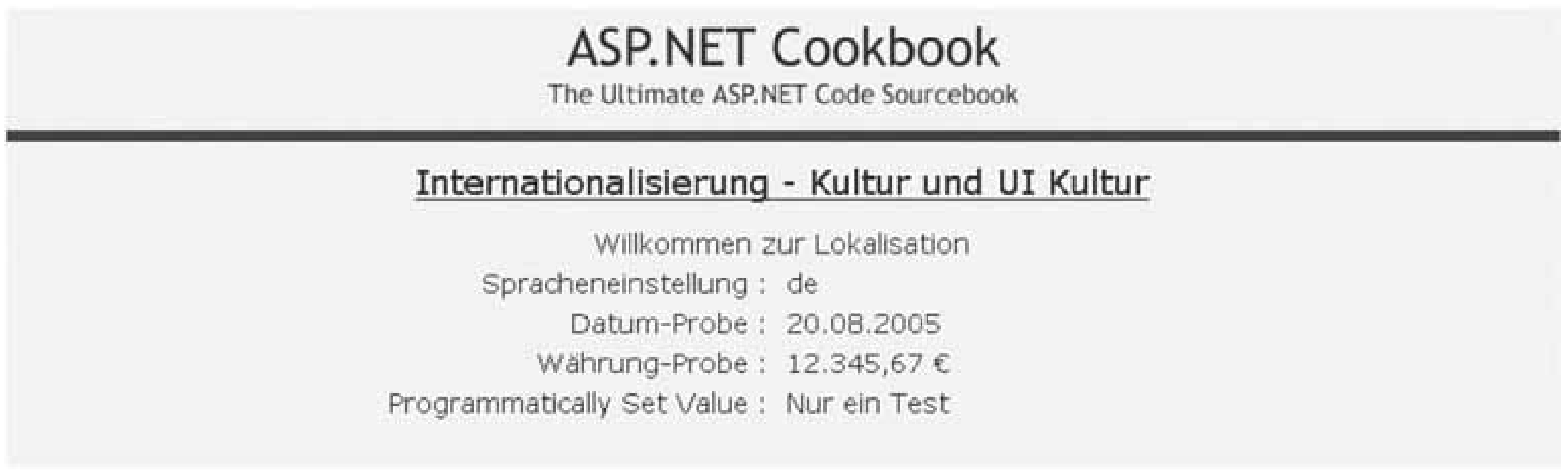


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Internationalization - Culture and UI Culture (VB)

Welcome to Localization
Language Setting : en-us
Date Sample: 12/7/2003
Currency Sample: $12,345.67

Figure 17-2. Multiple language outputGerman



ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Internationalisierung - Kultur und UI Kultur

Willkommen zur Lokalisation
Spracheneinstellung : de
Datum-Probe : 20.08.2005
Währung-Probe : 12.345,67 €
Programmatically Set Value : Nur ein Test

Discussion

ASP.NET 1.x provides extensive support for internationalizing applications; however, it requires you to w lot of code to set the text or values of every control that needs to be localized. In addition, the creation required resource files is tedious. ASP.NET 2.0 and Visual Studio 2005 have made the task of internatior an application much easier.

First, Visual Studio 2005 provides the ability to create the root resource file automatically for anASPX pag Resource files for a specific ASPX page are referred to as local resource files. These *.resx* files contain the localization information for a single ASPX page.

To create the root resource file, open the desired *.aspx* file, switch to Design mode, and then select Tool Generate Local Resources from the menu. Visual Studio will create a resource file and add an item for ea server control on the page. The resource file will be named the same as the page with a *.resx* extension a placed in the App_LocalResources folder of your application.

Second, during the process of creating the resource file, Visual Studio adds ameta:resourcekey attribute each of the server controls. This attribute is used by ASP.NET at runtime to set the value of the control automatically with the appropriate value from the resource file. You no longer need to provide code for e item on a page that must be localized. The only code you have to write is for setting the value of contro do not contain static text, such as dates, time, and other values that must be set programmatically.

In the example we have provided to demonstrate multiple language support, the ASPX page displays a we
message, the language setting from the browser request, the current date, a currency sample, and an
example of a control that is set programmatically, as shown in Figures 17-1(English) and 17-2 (German)
welcome message and the labels for each of the displayed items are set without writing any code.

The Page_Load method in the code-behind provides the code required to set the values of the controls. T
the Language Setting, we identify the user's preferred language from the collection of acceptable languag
returned with the page request. The language collection can contain zero or more languages, so we mus
ensure at least one language is in the collection. Because the language collection usually lists the langua
the order preferred by the user, selecting the first language in the list is generally acceptable.

In a production application, you may want to verify that your application supports the first language in the collection; if it does not, continue through the collection looking for a supported language.

Next, we set the values of the current date and sample currency. No special code is required. Set the va
you would in a single language application. The framework will take care of localizing the date and curre
values using the culture and UI culture for the user's preferred language.

Finally, we set the value of the Programmatically Set Value control to demonstrate how you can
programmatically retrieve an entry from the resource file. To retrieve a value from the local resource file
the GetLocalResourceObject method of the current page (MyClass or Me in VB, this in C#) passing the na
the required resource.

In ASP.NET 1.x, you had to handle the creation of the ResourceManager as well as setting the Culture and
Culture of the current thread to support a localized application. ASP.NET 2.0 does all of this work for you

The ResourceManager will handle all the dirty work of finding the resource file for the user's language, as
defaulting to the root resource file if a language is requested that is not supported by your application. I
to find the appropriate resource file because of a very strict naming convention used for the files.

The root resource file for local resources (resources for a specific page) is named the same as the page
.resx extension. In our example, the root resource is named *CH17InternationalCultureVB.aspx.resx* .

The names for additional language files must include the root name combined with the culture and subc
information in the format shown next, where *Rootname* is the root filename, *<languagecode>* defines the t
letter language code derived from ISO-639-1, and *<country/regioncode>* defines the two-letter country
region code from ISO-3166. The *languagecode* should be lowercase and the *country/regioncode* in upper

```
Rootname.<languagecode>-<country/regioncode>
```

For instance, the resource file containing text in German for our example is named
CH17InternationalCultureVB.aspx.resx.de-DE , where *de* indicates the German language and *DE* designa
the German localization of the language (as opposed to Austrian, Swiss, etc.).

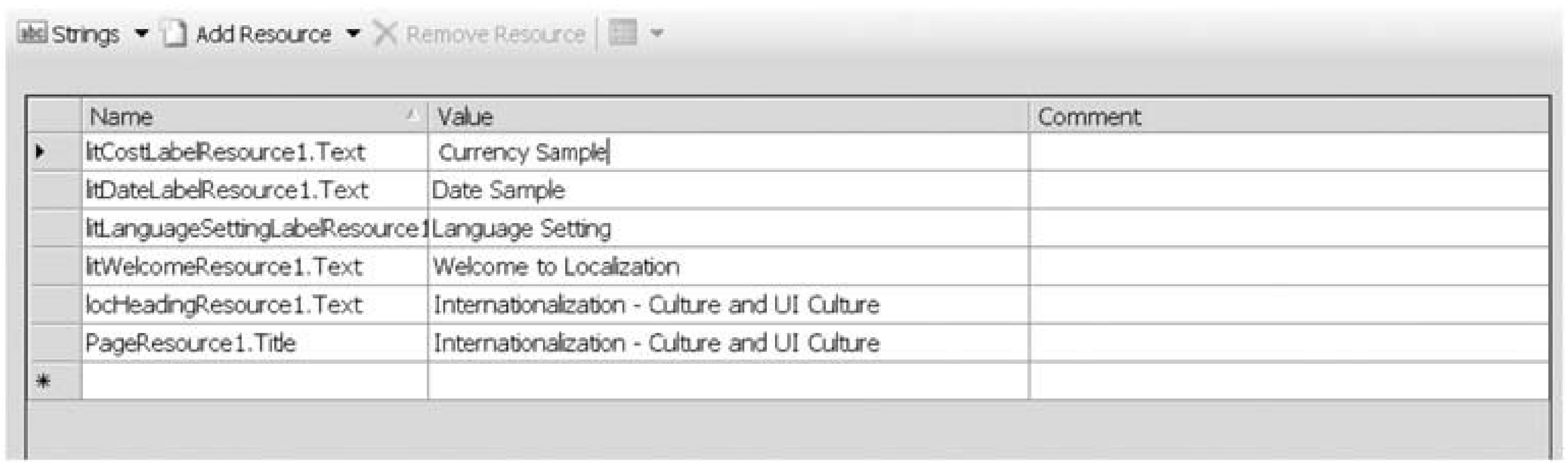
To find the required resource file, the resource manager uses the root name of the resource file and app
the UI Culture name to create the name of the resource file applicable to the user's language. The resou

manager then attempts to locate the resource file using a fairly complex set of rules. For our example, l keep it simple and say that if the specific resource file is not found on a first attempt, the root resource be used instead. (Refer to "Packaging and Deploying Resources" in the MSDN documentation for the full rules defining where the .NET runtime looks for resource files.)

Visual Studio 2005 provides a new resource file editor that makes creating and editing resource files eas in previous versions, providing the ability to manage image resources and string resources.

To add a local resource file, select the `App_LocalResources` folder in the Solution Explorer, right-click, sel `New Item` , and then select `Resource File` . The file must be named as described earlier. Figures 17-3and show the English and German resource files in the resource editor for our example.

Figure 17-3. English Resource File (CH17InternationalCultureVB.aspx.res



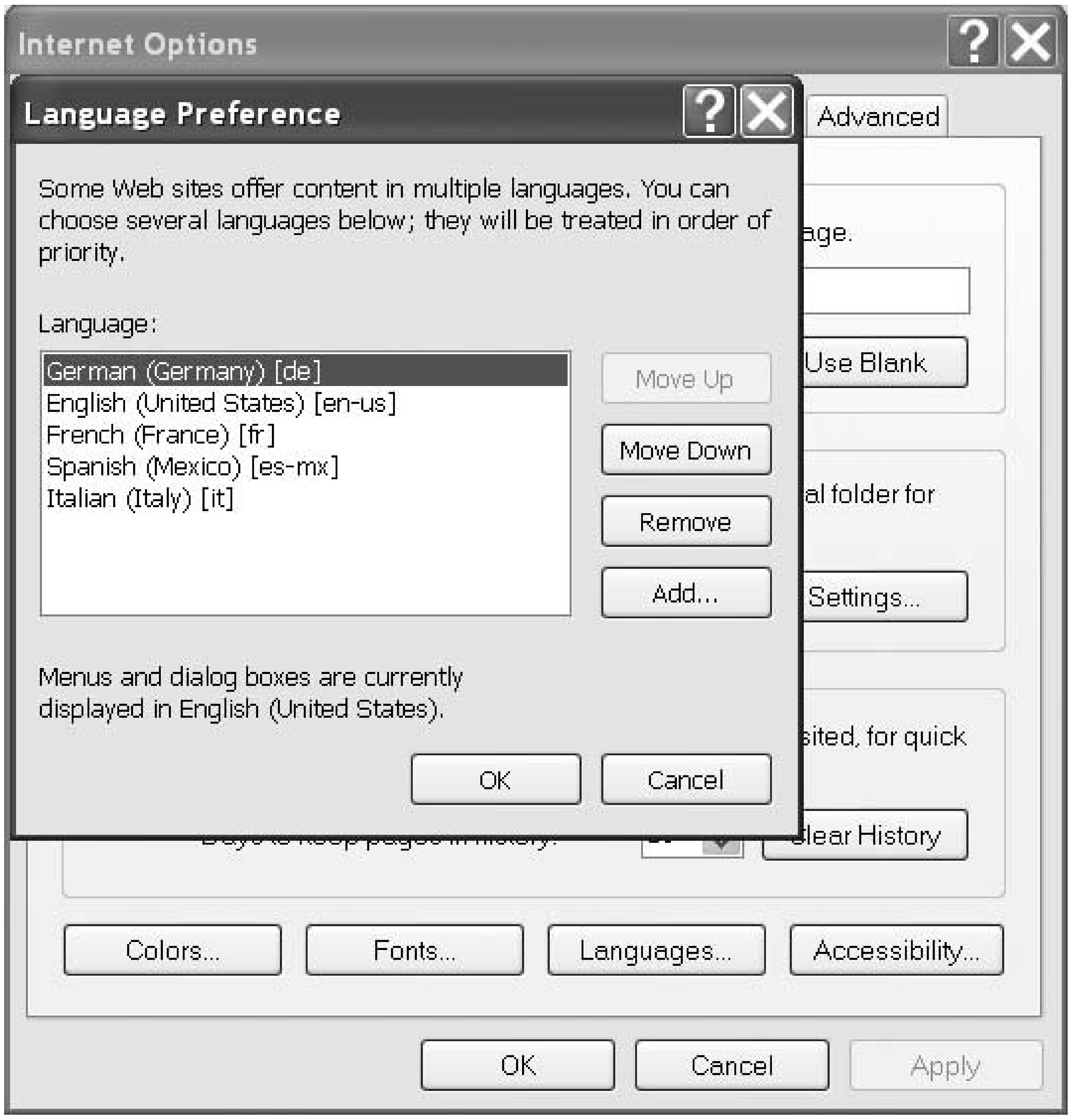
	Name	Value	Comment
▸	litCostLabelResource1.Text	Currency Sample	
	litDateLabelResource1.Text	Date Sample	
	litLanguageSettingLabelResource1	Language Setting	
	litWelcomeResource1.Text	Welcome to Localization	
	locHeadingResource1.Text	Internationalization - Culture and UI Culture	
	PageResource1.Title	Internationalization - Culture and UI Culture	
*			

Figure 17-4. German Resource File (CH17InternationalCultureVB.aspx.d DE.resx)

You can test your application for the supported languages by changing the preferred language in Internet Explorer. Select Tools Internet Options from the IE menu. Next, click the Languages button and the Language Preference dialog box will be displayed, as shown in Figure 17-5. Add the languages you need your application. To test a specific language, move it to the top of the list of languages.

Using resource files in an international application provides a cost-effective method for implementing the needed support for multiple languages. One thing to keep in mind when designing your application is the performance impact caused by "looking up" every string at runtime. This will result in longer rendering times for the pages. Your international application may require a more powerful web server than you would generally specify for a single language application.

Figure 17-5. Setting language preference in IE



Some Web sites offer content in multiple languages. You can choose several languages below; they will be treated in order of priority.

Language:

German (Germany) [de]
English (United States) [en-us]
French (France) [fr]
Spanish (Mexico) [es-mx]
Italian (Italy) [it]

Menus and dialog boxes are currently displayed in English (United States).

This example shows how to use local resource files for an individual page. In many applications, this res
duplication of resource data used on multiple pages. ASP.NET 2.0 supports global resources in addition t
resources. Recipe 17.3 provides an example of using global resources as well as overriding the`UI Cultur`
when currency must always be displayed in a specific currency format.

See Also

Recipe 17.3 and "Packaging and Deploying Resources" in the MSDN documentation for the full set of rul
defining where the .NET runtime looks for resource files

Example 17-1. Multiple language support (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
     AutoEventWireup="false"
   CodeFile="CH17InternationalCultureVB.aspx.vb"

    Inherits="ASPNetCookbook.VBExamples.CH17InternationalCultureVB"
  Title="Internationalization - Culture and UI Culture"
  meta:resourcekey="PageResource1" Culture="auto" UICulture="auto" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
     <div align="center" class="pageHeading">
          <asp:Localize ID="locHeading" runat="server"
                           meta:resourcekey="locHeadingResource1"
               Text="Internationalization - Culture and UI Culture (VB)" />
     </div>
  <table width="60%" align="center" border="0" class="labelText">
          <tr>
               <td align="center" colspan="2">
                    <asp:Literal id="litWelcome" Runat="server"
                                   Text="Welcome to Localization"
      meta:resourcekey="litWelcomeResource1" />

               </td>
          </tr>
    <tr>
               <td align="right" width="50%">
                    <asp:Literal id="litLanguageSettingLabel" Runat="server"
                                   meta:resourcekey="litLanguageSettingLabelResource1'
                    :&nbsp;
               </td>
               <td width="50%">
                    <asp:Literal id="litLanguageSetting" Runat="server"
                                   meta:resourcekey="litLanguageSettingResource1"  />
               </td>
          </tr>
    <tr>
               <td align="right" width="50%">
                    <asp:Literal id="litDateLabel" Runat="server"
                                   Text="Date Sample"
      meta:resourcekey="litDateLabelResource1" />
                    :&nbsp;
               </td>
      <td width="50%">
                    <asp:Literal id="litDate" Runat="server"
                              meta:resourcekey="litDateResource1"  />
               </td>
          </tr>
    <tr>
               <td align="right" width="50%">
                    <asp:Literal id="litCostLabel" Runat="server"
                                   Text="Currency Sample"
      meta:resourcekey="litCostLabelResource1" />
```

```
                   :&nbsp;
              </td>
      <td width="50%">
                   <asp:Literal id="litCost" Runat="server"
                   meta:resourcekey="litCostResource1"  />
              </td>
    </tr>

    <tr>
              <td align="right" width="50%">
                   <asp:Literal id="litProgrammaticallySetLabel" Runat="server"
                                    Text="Programmatically Set Value"
      meta:resourcekey="litProgrammaticallySetLabelResource1" />
                   :&nbsp;
              </td>
      <td width="50%">
                   <asp:Literal id="litProgrammaticallySet" Runat="server" />
              </td>
    </tr>
  </table>
</asp:Content>
```

Example 17-2. Multiple language support code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code-behind for
  '''  CH17InternationalCultureVB.aspx
  ''' </summary>
  Partial Class CH17InternationalCultureVB
    Inherits System.Web.UI.Page
 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Private Sub Page_Load(ByVal sender As Object, _
                          ByVal e As System.EventArgs) Handles Me.Load
      Const sampleValue As Single = 12345.67
```

```
   'set the control displaying the browser culture setting
   If ((Not IsNothing(Request.UserLanguages)) AndAlso _
          (Request.UserLanguages.Length > 0)) Then
        litLanguageSetting.Text = Request.UserLanguages(0)
      Else
        litLanguageSetting.Text = "None"
      End If

      'set the sample date to the current date
   litDate.Text = DateTime.Now.ToShortDateString()

   'set the sample currency value
   litCost.Text = sampleValue.ToString("C")

   'set a control programmatically from the local resource file
   litProgrammaticallySet.Text = _
     CStr(MyClass.GetLocalResourceObject("ProgrammaticallySetValue"))
    End Sub 'Page_Load
  End Class 'CH17InternationalCultureVB
End Namespace
```

Example 17-3. Multiple language support code-behind (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code-behind for
  /// CH17InternationalCultureCS.aspx
  /// </summary>
  public partial class CH17InternationalCultureCS : System.Web.UI.Page
  {
    ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
      const double sampleValue = 12345.67;

   // set the control displaying the browser culture setting
   if ((Request.UserLanguages != null) &&
          (Request.UserLanguages.Length > 0))
      {
        litLanguageSetting.Text = Request.UserLanguages[0];
      }
   else
   {
        litLanguageSetting.Text = "None";
      }

   // set the sample date to the current date
   litDate.Text = DateTime.Now.ToShortDateString();

   // set the sample currency value
   litCost.Text = sampleValue.ToString("C");

   // set a control programmatically from the local resource file
   litProgrammaticallySet.Text =
        (String)(this.GetLocalResourceObject("ProgrammaticallySetValue"));
   } // Page_Load
  } // CH17InternationalCultureCS
}
```

Example 17-4. English resource file

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
     <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
 <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
   <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
     <xsd:attribute name="type" type="xsd:string" />
     <xsd:attribute name="mimetype" type="xsd:string" />
     <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
            <xsd:element name="assembly">
              <xsd:complexType>
                <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="name" type="xsd:string" />
```

```
             </xsd:complexType>
           </xsd:element>
           <xsd:element name="data">
             <xsd:complexType>
               <xsd:sequence>
                 <xsd:element name="value" type="xsd:string" minOccurs="0"
                                 msdata:Ordinal="1" />
                 <xsd:element name="comment" type="xsd:string" minOccurs="0"
                                 msdata:Ordinal="2" />
               </xsd:sequence>
               <xsd:attribute name="name" type="xsd:string" use="required"
                                 msdata:Ordinal="1" />
               <xsd:attribute name="type" type="xsd:string"
                                 msdata:Ordinal="3" />
               <xsd:attribute name="mimetype" type="xsd:string"
                                 msdata:Ordinal="4" />
               <xsd:attribute ref="xml:space" />
             </xsd:complexType>
           </xsd:element>
  <xsd:element name="resheader">

             <xsd:complexType>
               <xsd:sequence>
                 <xsd:element name="value" type="xsd:string" minOccurs="0"
                                 msdata:Ordinal="1" />
               </xsd:sequence>
               <xsd:attribute name="name" type="xsd:string" use="required" />
             </xsd:complexType>
           </xsd:element>
         </xsd:choice>
       </xsd:complexType>
     </xsd:element>
   </xsd:schema>
   <resheader name="resmimetype">
     <value>text/microsoft-resx</value>
   </resheader>
   <resheader name="version">
      <value>2.0</value>
</resheader>
<resheader name="reader">
     <value>System.Resources.ResXResourceReader, System.Windows.Forms,
              Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
     </value>
   </resheader>
<resheader name="writer">
     <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
              Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
     </value>
   </resheader>
    <data name="litCostLabelResource1.Text" xml:space="preserve">
     <value>Currency Sample</value>
   </data>
```

```
     <data name="litDateLabelResource1.Text" xml:space="preserve">
      <value>Date Sample</value>
    </data>
    <data  name="litLanguageSettingLabelResource1.Text"  xml:space="preserve">
      <value>Language Setting</value>
    </data>
     <data name="litProgrammaticallySetLabelResource1" xml:space="preserve">
      <value>Programmatically Set</value>
 </data>
 <data  name="litWelcomeResource1.Text"  xml:space="preserve">
      <value>Welcome to Localization</value>
 </data>
 <data  name="locHeadingResource1.Text"  xml:space="preserve">
   <value>Internationalization - Culture and UI Culture (VB)</value>
 </data>
 <data  name="PageResource1.Title"  xml:space="preserve">
      <value>Internationalization - Culture and UI Culture</value>
 </data>
    <data name="ProgrammaticallySetValue" xml:space="preserve">
      <value>Just a Test</value>
    </data>
</root>
```

Example 17-5. German resource file

```
<?xml version="1.0" encoding="utf-8"?>
<root>
   <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                 xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
     <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
     <xsd:attribute name="name" use="required" type="xsd:string" />
     <xsd:attribute name="type" type="xsd:string" />
     <xsd:attribute name="mimetype" type="xsd:string" />
     <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
    <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
     <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
    <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
                                 msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
                                 msdata:Ordinal="2" />
              </xsd:sequence>
     <xsd:attribute name="name" type="xsd:string" use="required"
                                 msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string"
                                 msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string"
                                 msdata:Ordinal="4" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
    <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
                                 msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>

            </xsd:complexType>
       </xsd:element>
    </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
     <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms,
             Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
    </value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
             Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
    </value>
  </resheader>
   <data  name="litCostLabelResource1.Text"  xml:space="preserve">
    <value>Währung-Probe</value>
   </data>
   <data  name="litDateLabelResource1.Text"  xml:space="preserve">
     <value>Datum-Probe</value>
  </data>
  <data  name="litLanguageSettingLabelResource1.Text"  xml:space="preserve">
     <value>Spracheneinstellung</value>
   </data> <data  name="litProgrammaticallySetLabelResource1"  xml:space="preserve">
     <value>Programmatically Satz</value>
   </data>
   <data  name="litWelcomeResource1.Text"  xml:space="preserve">
     <value>Willkommen zur Lokalisation</value>
   </data>
   <data  name="locHeadingResource1.Text"  xml:space="preserve">
     <value>Internationalisierung - Kultur und UI Kultur</value>
   </data>
   <data  name="PageResource1.Title"  xml:space="preserve">
     <value>Internationalisierung - Kultur und UI Kultur</value>
   </data>
  <data name="ProgrammaticallySetValue" xml:space="preserve">
     <value>Nur ein Test</value>
  </data>
</root>
```

PREV

Recipe 17.4. Using Global Resources and Overriding Currency Formatting

Problem

You want to reuse resources throughout your international application, and you must always display currency with the U.S. dollar symbol, but all other text must follow the language setting in the browser.

Solution

Create a root global resource for your application, add the required items, explicitly set the values of static text in the *.aspx* files, set the culture and `uiCulture` attributes of the `<globalization>` element in *web.config* to `auto`, and set the values for dynamic controls in the code-behind class overriding the `UI Culture` when setting currency values.

Create a global resource file as follows:

- Create an `App_GlobalResources` folder in the root of your application.
- Add a resource file to the `App_GlobalResources` folder.
- Edit the resource file to include the required items.
- Duplicate the root resource file for each language and culture to be supported by your application, setting the values to the resource items as appropriate for the language.

In the *.aspx* files of your application:

- Explicitly set the values for all controls that display static text.

Set the `culture` and `uiCulture` attributes of the `<globalization>` element in *web.config* to `auto`;

```
<globalization culture="auto" uiCulture="auto" />
```

In the code-behind class for each page that needs to support multiple languages, use the .NET language of your choice to:

- Set the values of controls used for date.
- Set the values of controls used for currency overriding the UI Culture to be en-US.
- Set the text of any controls that need to be set programmatically.

Examples 17-7, 17-8 through 17-9 show the *.aspx* file and VB and C# code-behind files for an application we've written to demonstrate this solution. Example 17-6 shows the *web.config* settings for globalization.

Discussion

It is not uncommon to have an international application that needs to use the sameresources in multiple pages and needs to display dates, a currency value, or other data in a format specific to a language or culture that is not the default. For example, you might need to display currency values ir U.S. dollars but all text and dates in the local language of the user. ASP.NET provides support for both requirements.

Global resources are resources that can be used throughout an application (see Recipe 17.2 for an example of using local resources). They are identical in format to local resource files but must be created manually and are placed in the `App_GlobalResources` folder of your application. Though they must be created manually, the resource editor in Visual Studio 2005 simplifies the task.

Like local resources, the data from global resources can be used to set the values of controls in your `ASPX` pages without requiring you to write any code. The technique used is different, however. Instead of using `meta:resourcekey` attributes in the server controls, you explicitly set the properties of the controls.

```
<asp:Literal id="litLanguageSettingLabel" Runat="server"
     Text="<%$ Resources: ASPNetCookbook2, WelcomeToLocalizationResource %>" />
```

The syntax for the command is shown below, where *`<Root Resource Filename>`* is the name of the root global resource file and *`<Resource ID>`* is the ID of the specific resource item in the file.

```
<%$ Resources: <Root Resource Filename>, <Resource ID> %>
```

To override the formatting of currency values to be displayed in a specific format, such as U.S. dollars, you must set the format provider to a `CultureInfo` object that is in turn set to the desired `UI Culture`, as shown below. If you need to set currency values in many places in your application, you should implement a single method that applies the setting. This will make changes simpler to carry out when they are required.

```
litCost.Text = sampleValue.ToString("C", _
                                                 New CultureInfo("en-US"))
```

C#

```
    litCost.Text = sampleValue.ToString("C",
                                                 new CultureInfo("en-US"));
```

When you need to set values programmatically, global resources have a significant advantage over local resources. The global resources are strongly typed, and Intellisense is fully supported. Setting a value from a global resource is identical to using the property of a class, as shown below, where the name of our root global resource file is `ASPNetCookbook2` and the name of the resource item is `ProgrammaticallySetValue`:

VB

```
 litProgrammaticallySet.Text = _
       Resources.ASPNetCookbook2.ProgrammaticallySetValue
```

C#

```
    litProgrammaticallySet.Text =
       Resources.ASPNetCookbook2.ProgrammaticallySetValue;
```

This recipe and Recipe 17.2 have only touched on the features available in Visual Studio 2005 and ASP.NET 2.0 for internationalizing an application. Search for "ASP.NET Web Page Resources Overview" in the MSDN Library for additional information.

See Also

Recipe 17.2 and "ASP.NET Web Page Resources Overview" in the MSDN Library

Example 17-6. web.config settings for globalization

```
<?xml version="1.0"?>
<configuration>
  <system.web>

    …

 <!--  GLOBALIZATION
          This section sets the globalization settings of the application.
    -->
 <globalization culture="auto" uiCulture="auto" />
  </system.web>
</configuration>
```

Example 17-7. Global Resources and Overriding Currency (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH17InternationalOverridingCultureVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH17InternationalOverridingCultureVB"
  Title="<%$ Resources: ASPNetCookbook2, HeadingResource %>" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">
    <asp:Localize ID="locHeading" runat="server"
      Text="<%$ Resources: ASPNetCookbook2, HeadingResource %>"   />
  </div>
   <table width="60%" align="center" border="0" class="labelText">
    <tr>
       <td align="center" colspan="2">
        <asp:Literal id="litWelcome" Runat="server"
     Text="<%$ Resources: ASPNetCookbook2, WelcomeToLocalizationResource %>" />
      </td>
    </tr>
 <tr>
       <td align="right" width="50%">
        <asp:Literal id="litLanguageSettingLabel" Runat="server"
     Text="<%$ Resources: ASPNetCookbook2, WelcomeToLocalizationResource %>" />
         :&nbsp;
      </td>
   <td width="50%">
        <asp:Literal id="litLanguageSetting" Runat="server" />
      </td>
    </tr>

 <tr>
       <td align="right" width="50%">
        <asp:Literal id="litDateLabel" Runat="server"
     Text="<%$ Resources: ASPNetCookbook2, SampleDateLabel %>"   />
         :&nbsp;
      </td>
   <td width="50%">
        <asp:Literal id="litDate" Runat="server" />
      </td>
    </tr>
 <tr>
      <td align="right" width="50%">
        <asp:Literal id="litCostLabel" Runat="server"
     Text="<%$ Resources: ASPNetCookbook2, SampleCostLabelResource %>"   />
         :&nbsp;
      </td>
   <td width="50%">
        <asp:Literal id="litCost" Runat="server" />
      </td>
    </tr>
 <tr>
      <td align="right" width="50%">
        <asp:Literal id="litProgrammaticallySetLabel" Runat="server"
     Text="<%$ Resources: ASPNetCookbook2, ProgrammaticallySetLabel %>"   />
         :&nbsp;
      </td>
   <td width="50%">
        <asp:Literal id="litProgrammaticallySet" Runat="server" />
      </td>
    </tr>
  </table>
</asp:Content>
```

Example 17-8. Global Resources and Overriding Currency (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Globalization

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code-behind for
  ''' CH17InternationalOverridingCultureVB.aspx
  ''' </summary>
  Partial Class CH17InternationalOverridingCultureVB
    Inherits System.Web.UI.Page
 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.

 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Private Sub Page_Load(ByVal sender As Object, _
                         ByVal e As System.EventArgs) Handles Me.Load
      Const sampleValue As Single = 12345.67

   'set the control displaying the browser culture setting
   If ((Not IsNothing(Request.UserLanguages)) AndAlso _
          (Request.UserLanguages.Length > 0)) Then
        litLanguageSetting.Text = Request.UserLanguages(0)
      Else
```

```
        litLanguageSetting.Text = "None"
      End If

      'set the sample date to the current date
   litDate.Text = DateTime.Now.ToShortDateString()

   'set the sample currency value forcing US formatting
   litCost.Text = sampleValue.ToString("C", _
                                       New CultureInfo("en-US"))

      'set a control programmatically from the global resource file
   litProgrammaticallySet.Text = _
        Resources.ASPNetCookbook2.ProgrammaticallySetValue
    End Sub 'Page_Load
  End Class 'CH17InternationalOverridingCultureVB
End Namespace
```

Example 17-9. Global Resources and Overriding Currency (.cs)

```
using System;
using System.Globalization;

namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code-behind for
  ///  CH17InternationalOverridingCultureCS.aspx
  /// </summary>
  public partial class CH17InternationalOverridingCultureCS :
    System.Web.UI.Page
  {
    ///*************************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>

 protected void Page_Load(object sender, EventArgs e)
    {
      const double sampleValue = 12345.67;

   // set the control displaying the browser culture setting
   if ((Request.UserLanguages != null) &&
       (Request.UserLanguages.Length > 0))
      {
        litLanguageSetting.Text = Request.UserLanguages[0];
      }
   else
      {
        litLanguageSetting.Text = "None";
      }

   // set the sample date to the current date
   litDate.Text = DateTime.Now.ToShortDateString();

   // set the sample currency value forcing US formatting
   litCost.Text = sampleValue.ToString("C",
                                       new CultureInfo("en-US"));

      // set a control programmatically from the global resource file
   litProgrammaticallySet.Text =
        Resources.ASPNetCookbook2.ProgrammaticallySetValue;
    } // Page_Load
  } // CH17InternationalOverridingCultureCS
}
```

Chapter 18. File Operations

PREV

18.0 Introduction

Downloading a file from a server is a common requirement of web applications, whereas uploading a file is less so. For example, you might want your application to allow users to download PDF, binary, or image files to their browsers. Alternatively, you might have a content management system to which you want to let users upload images or binary files.

ASP.NET provides an easy-to-use and flexible infrastructure for completing either task. You can download a file to the browser for display, storage, or printing by streaming it to the `Response` object, a simple task in ASP.NET.

Uploading a file to the web server for storage is straightforward, particularly with the `FileUpload` control introduced with ASP.NET 2.0.

Sometimes you may want to upload a file to the server for processing only, without storing it there, and at other times you may need to store the contents of an uploaded file to a database. For example, you might want to do the former to avoid having to deal with problems associated with files being uploaded with the same names, inadvertently filling the hard drive, or allowing the ASP.NET write privileges on the local filesystem. Storing an uploaded file in a database is useful when you want to keep a complete record of the file set apart from the web server's files system. The recipes in this chapter show you how to do all of these things.

Recipe 18.2. Downloading a File from the Web Server

Problem

You need to provide the ability for a user to download a file from the web server.

Solution

Use the `Directory` and `FileInfo` classes to gather and present the names of the files you want to make available for download to the user. Display their names in a `ListBox` with a button to initiate the download. When the user clicks the button, stream the selected file to the browser.

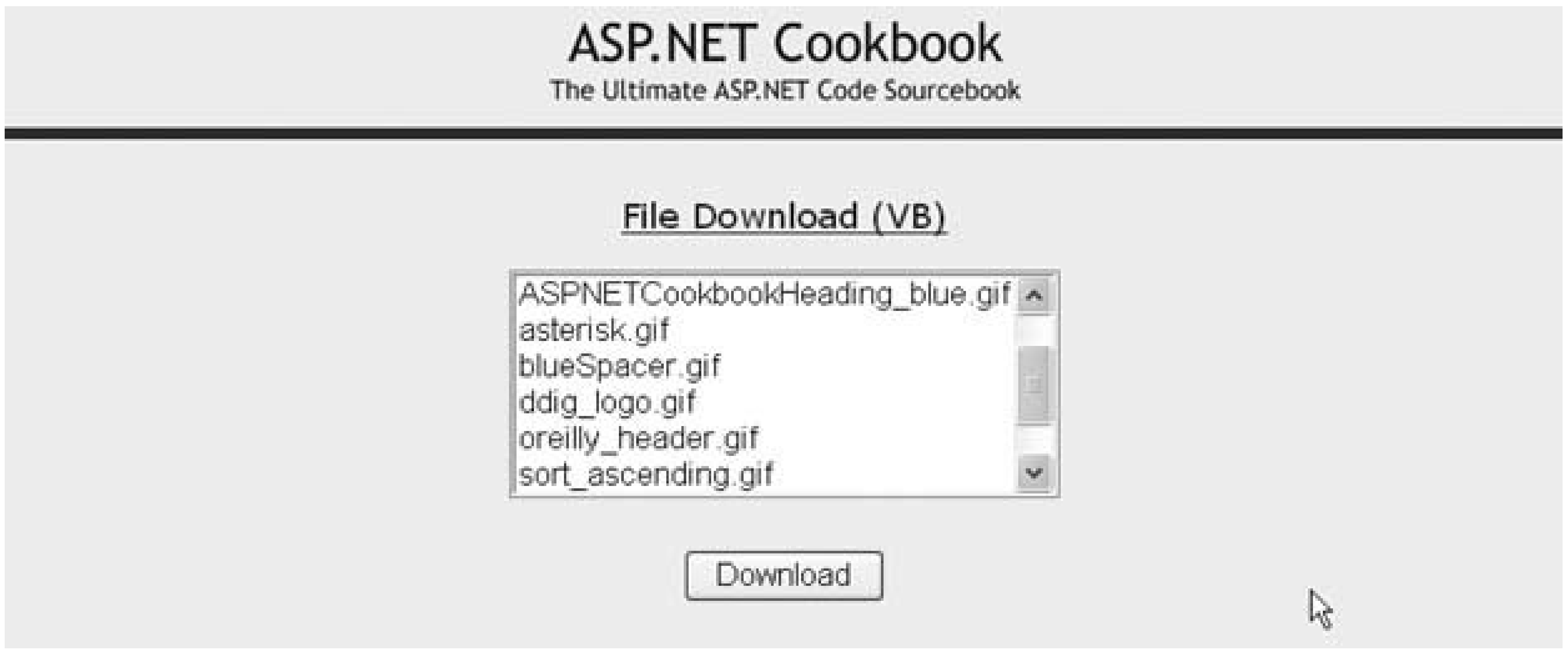
In the *.aspx* file, add a `ListBox` and a Download (or equivalently named) button.

In the code-behind class for the page, use the .NET language of your choice to:

1. Create a list of available files to download using the `GetFiles` method of the `Directory` class.
2. Populate the `ListBox` with the filenames by binding the list of files to the `ListBox`.
3. Process the Download button click event and stream the selected file to the browser using the `Response` object.

Examples 18-1, 18-2 through 18-3 show the *.aspx* file and VB and C# code-behind files for an application that illustrates our solution by populating the `ListBox` with a list of files located in the application's images directory. The populated `ListBox` is shown in Figure 18-1, and the prompt that is output when the user selects a file and clicks the Download button is shown in Figure 18-2.

Figure 18-1. Listing files to be downloaded

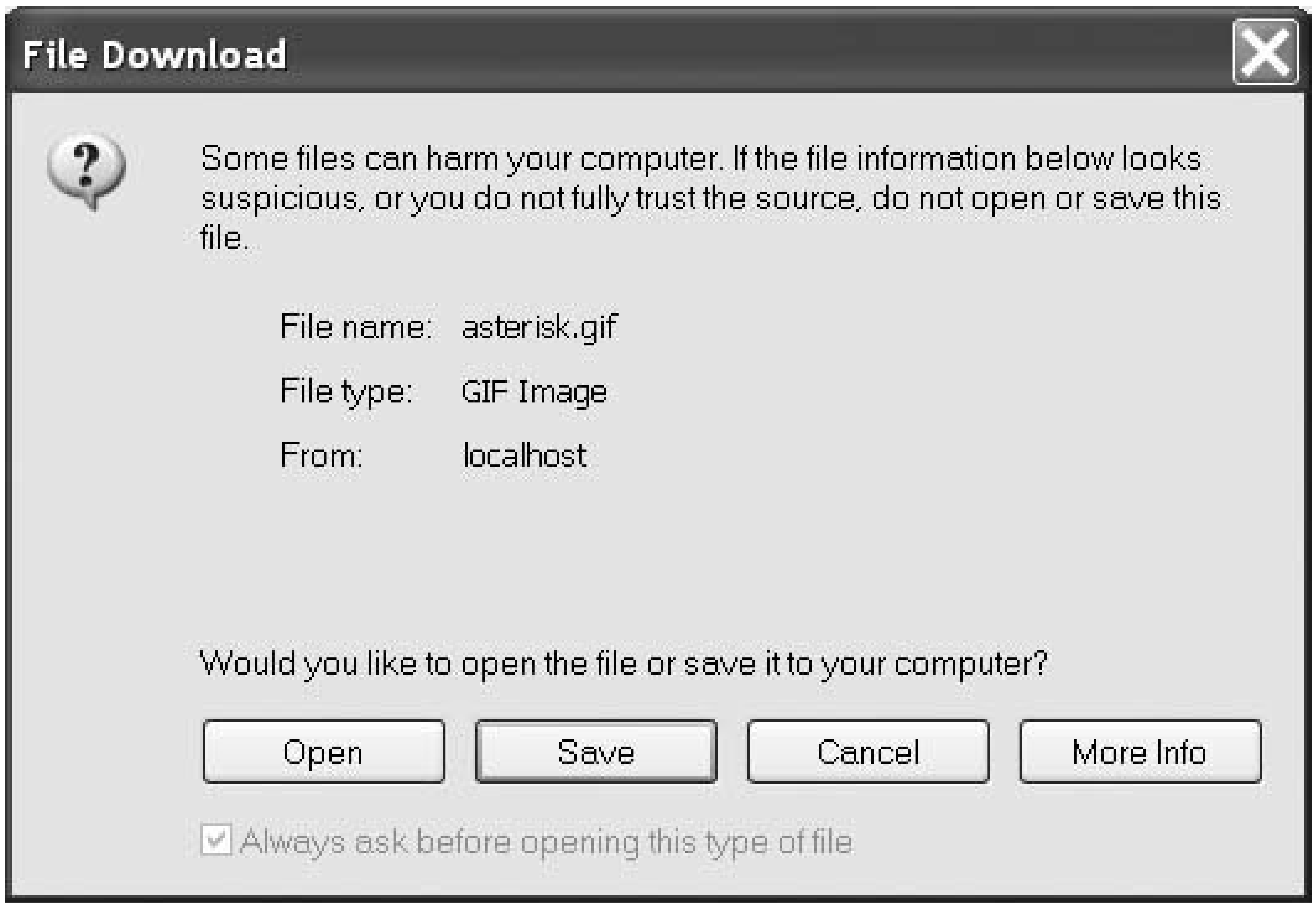


Discussion

Downloading a file to the browser for display, storage, or printing is a common requirement of a web application. PDF and Word files are perhaps the most commonly downloaded file types though image audio, video, and text files are common also.

Downloading a file from a server is a two-step process. The first step is to gather and present to the user a list of the available files that can be downloaded, along with a button to initiate the download. The second step is to process the button click event and stream the selected file to the browser.

Figure 18-2. File download user prompt


File Download

Some files can harm your computer. If the file information below looks suspicious, or you do not fully trust the source, do not open or save this file.

File name: asterisk.gif

File type: GIF Image

From: localhost

Would you like to open the file or save it to your computer?

In our example, we use a ListBox to present a list of available files to the user. The list is populated in the Page_Load method of the code-behind with a list of files located in the images directory of the application. (In a production application, you may want to create a download folder instead.)

To populate the ListBox, we use the GetFiles method of the Directory class to gather the fully qualified filenames of the files in the specified folder and return them as an array. TheGetFiles method returns a fully qualified filename for each file it finds, so our code needs to remove the path information for each file to simplify the list we present to the user.

Next, we bind the files array to the ListBox and select the first entry in the list.

Selecting the first entry in the list will simplify the code because no validation is required if we can always assume an item is selected.

When the user clicks the Download button to initiate the download, thebtnDownload_ServerClick method in the code-behind executes. In this routine, we use theMapPath method of the Server class to create a fully qualified filename, which we use to instantiate aFileInfo object to provide easy access to the length of the file needed for the download.

To stream a file to a browser, you must write it to theResponse object. The first step in writing a file to the Response object is to call its Clear method to remove any data currently in the buffer stream. If the Response object contains data and you attempt to write a file to it, you will receive a corrupted file error.

Before writing the file, use the AddHeader method of the Response object to add the name of the file being downloaded and its length to the output stream. You must use the ContentType method to specify the content type of the file. In this example, the type is set to application/octet-stream so the browser will treat the output stream as a binary stream and prompt the user to select a location to which to save the file. In your own application you may want to set the content type to an explicit file type, such as application/PDF or application/msword. Setting the content type to the explicit file type allows the browser to open it with the application defined to handle the specified file type on the client machine.

Now you are ready to write the file to the Response object using Response.WriteFile. When the operation is complete, call Response.End to send the file to the browser.

Things to remember when downloading files:

- Any code that appears after the Response.End statement will not be executed. The Response.End statement sends all buffered data in the Response object to the client, stops the execution of the page, and raises the Application_EndRequest event. For more information on the application behavior when calling Response.End, refer to Knowledge Base article KB312629.
- The only data that can be included in the Response stream is from the file to download. If any other data is included, the downloaded file will be corrupted. This may occur if your application uses the Application_EndRequest method to append a footer to all pages.

In our example, a list of files currently residing on the filesystem is presented to the user to select from and download. If your application is going to create the file dynamically, it will be unnecessary to present a list or to save the file to the filesystem. Instead, remove the ListBox and add whatever controls are needed to collect the information you need to dynamically create the file. When the user clicks Download, create your file and generate a byte array containing the file data. Instead of using the WriteFile method of the Response object, use the BinaryWrite method, as shown here:

```
Response.BinaryWrite([your byte array])

Response.BinaryWrite([your byte array]);
```

For another approach to implementing the solution described in this example using an HTTP handler, refer to Recipe 20.2.

See Also

Recipe 20.2 for implementing file downloads with an HTTP handler; Knowledge Base article KB312629 for more information on the system operation when Response.End, Response.Redirect, and

Server.Transfer are called

Example 18-1. Downloading a file (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH18FileDownloadVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH18FileDownloadVB"
 Title="File Download" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  File Download (VB)
 </div>
 <table width="90%" align="center" border="0">
  <tr>
   <td align="center">
    <asp:ListBox ID="lstFiles" Runat="server" Rows="6" />
   </td>
  </tr>
  <tr>
   <td align="center">
    <br />
    <input id="btnDownload" runat="server"
       type="button"
       value="Download"
       onserverclick="btnDownload_ServerClick" />
    </td>
  </tr>
 </table>
</asp:Content>
```

Example 18-2. Downloading a file code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System.IO

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH18FileDownloadVB.aspx
 ''' </summary>
 Partial Class CH18FileDownloadVB
```

```
Inherits System.Web.UI.Page
 '''*************************************************************************
''' <summary>
''' This routine provides the event handler for the page load event. It
''' is responsible for initializing the controls on the page.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
 Dim files() As String
 Dim index As Integer

 If (Not Page.IsPostBack) Then
  'get list of files in the images directory (just for example here)
  files = Directory.GetFiles (Server.MapPath("images"))

  'for display purposes, remove the path to the file
  For index = 0 To files.Length - 1
  files(index) = New FileInfo(files(index)).Name
  Next index

  'bind the list of files to the ListBox on the form
  lstFiles.DataSource = files
  lstFiles.DataBind()

  'select the first entry in the list
  'NOTE: This is done to simplify the example since preselecting an
  '    item eliminates the needs to verify an item was selected
  lstFiles.SelectedIndex = 0
 End If
End Sub 'Page_Load

 '''*************************************************************************
''' <summary>
''' This routine provides the event handler for the download button click
''' event. It is responsible for reading the selected file from the file
''' system and streaming it to the browser.
''' </summary>
'''
''' <param name="sender">Set to the sender of the event</param>
''' <param name="e">Set to the event arguments</param>
Protected Sub btnDownload_ServerClick(ByVal sender As Object, _
             ByVal e As System.EventArgs)
 Dim file As FileInfo
 Dim filename As String

 'get the fully qualified name of the selected file
 filename = Server.MapPath("images") & "\" & _
     lstFiles.SelectedItem.Text
    'get the file data since the length is required for the download
    file = New FileInfo(filename)
    'write it to the browser
    Response.Clear()
    Response.AddHeader("Content-Disposition", _
        "attachment; filename=" & lstFiles.SelectedItem.Text)
    Response.AddHeader("Content-Length", _
                            file.Length.ToString())
    Response.ContentType = "application/octet-stream"
    Response.WriteFile(filename)
    Response.End()
  End Sub 'btnDownload_ServerClick
 End Class 'CH18FileDownloadVB
End Namespace
```

Example 18-3. Downloading a file code-behind (.cs)

```
using System;
using System.IO;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH18FileDownloadCS.aspx
 /// </summary>
 public partial class CH18FileDownloadCS : System.Web.UI.Page
 {
  ///***********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   string[] files = null;
   int index;

   if (!Page.IsPostBack)
   {
    // get list of files in the images directory
    files = Directory.GetFiles(Server.MapPath("images"));

    // for display purposes, remove the path to the file
```

```
      for (index = 0; index < files.Length; index++)
      {
      files[index] = new FileInfo(files[index]).Name;
      }
      // bind the list of files to the ListBox on the form
      lstFiles.DataSource = files;
      lstFiles.DataBind();

      // select the first entry in the list
      // NOTE: This is done to simplify the example since preselecting an
      //       item eliminates the needs to verify an item was selected
      lstFiles.SelectedIndex = 0;
     }
    } // Page_Load

    ///***********************************************************************
    /// <summary>
    /// This routine provides the event handler for the download button click
    /// event. It is responsible for reading the selected file from the file
    /// system and streaming it to the browser.
    /// </summary>
    ///
    /// <param name="sender">Set to the sender of the event</param>
    /// <param name="e">Set to the event arguments</param>
    protected void btnDownload_ServerClick(Object sender,
                        System.EventArgs e)
    {
     FileInfo file;
     String filename;

     //get the fully qualified name of the selected file
     filename = Server.MapPath("images") + "\\" +
         lstFiles.SelectedItem.Text;

     // get the file data since the length is required for the download
     file = new FileInfo(filename);

     // write it to the browser
     Response.Clear();
     Response.AddHeader("Content-Disposition",
         "attachment; filename=" + lstFiles.SelectedItem.Text);
     Response.AddHeader("Content-Length",
                        file.Length.ToString());
     Response.ContentType = "application/octet-stream";
     Response.WriteFile(filename);
     Response.End();
    } // btnDownload_ServerClick
   } // CH18FileDownloadCS
  }
```

PREV

PREV

Recipe 18.3. Uploading a File to the Web Server

Problem

You need to provide the ability for a user to upload a file to the web server filesystem.

Solution

Add to your page a `FileUpload` control and a button to initiate the upload process. When the user clicks the button, the code-behind can save the file to the filesystem.

Create a folder within your application file structure where the uploaded files will be placed, such as as one named *uploads*.

In the *.aspx* file:

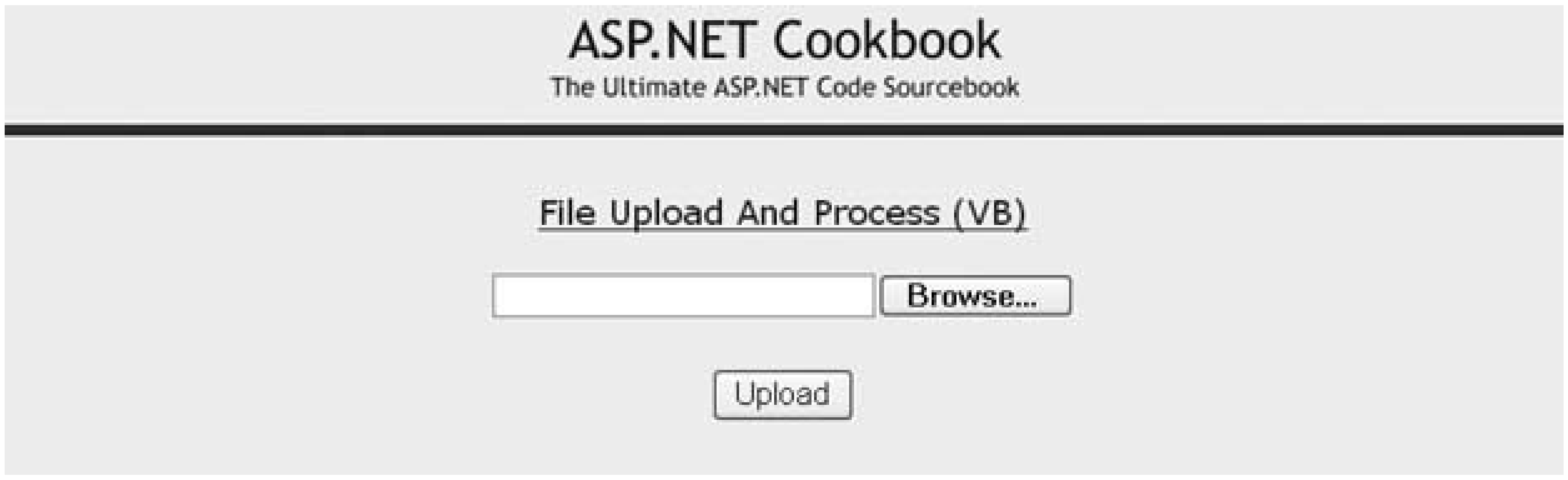
1. Add a `FileUpload` control.
2. Add an Upload (or equivalently named) button.

In the code-behind class for the page, use the .NET language of your choice to:

1. Verify, in the Upload button click event handler, that the file content has been uploaded by checking the `HasFile` property of the `FileUpload` control.
2. Save the file to the local filesystem on the server using the`SaveAs` method of the `FileUpload` object.

Example 18-4 shows the *.aspx* file for an application we've written to demonstrate this solution by allowing you to browse for a file and uploading the chosen file to your web server's local filesystem when you click the Upload button. The code-behind files for the application are shown inExamples 18-5 (VB) and 18-6 (C#). The UI for uploading a file is shown in Figure 18-3.

Figure 18-3. UI for uploading a file


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
File Upload And Process (VB)
Browse...
Upload

Discussion

Most applications do not require uploading files to a web server. Nevertheless, here are a few examples of applications for which this capability comes in handy:

Departmental content management system

Uploading images or documents

Technical support site

Uploading error logs and defective documents or files

Graphics library

Allowing users to be able to make their own graphics file submissions

When uploading files is needed, the support provided by ASP.NET makes the implementation straightforward.

The application we've written to illustrate the solution allows you to browse for a file and upload the chosen file to your web server's local filesystem when you click the Upload button. For all that it accomplishes, the application requires a remarkably small amount of code.

The *.aspx* file must also include a `FileUpload` control. This causes the browser to render the `input` element with a Browse... button to allow the user to browse to the file to upload.

To give your user the ability to initiate the upload, you'll want to add an Upload button to your *.aspx* file. Your *.aspx* file can contain as many controls as you need to allow users to interact with the application, such as other input controls or dropdowns. The user's data will be submitted like any other form for use on the server side.

Place code to save the uploaded file to the server in the Upload button click event handler of the code-behind. Before saving the file, confirm that the file upload completed successfully. This can be done by checking the `HasFile` property of the `FileUpload` control to ensure it is set to `true`. A production application should use validation controls to check this condition and output an error message to the user. Refer to Chapter 3 for a discussion of validation controls.

After you verify the file has been uploaded, it can be saved to the filesystem. The `SaveAs` method of the `FileUpload` object saves the uploaded file contents to a file on the web server. It requires a fully qualified filename on the web server. As shown in the example, we get the name of the file from the `FileName` property of the `FileUpload` control and then build a fully qualified filename for the storage location and name on the web server.

By default, the account under which ASP.NET runs does not have permissionto write files to the filesystem of the server. The account used varies with the server on which your application runs and depends on whether your application uses impersonation.

The name of the ASP.NET user can be determined by creating an ASP.NET page with the following content and displaying the page in a browser:

```
<%@ Page Language="VB" %>
<% Response.Write(System.Security.Principal.WindowsIdentity.
GetCurrent().Name) %>

<%@ Page Language="C#" %>
<%Response.Write(System.Security.Principal.WindowsIdentity.
GetCurrent().Name); %>
```

You will need to modify the security settings on the folder used for uploads to allow the account used by ASP.NET to write to the folder. The steps for doing so vary somewhat for the different flavors of Windows, but the basic steps are these:

1. Using Windows Explorer, browse to the folder where the uploaded files will be saved.
2. Access the security settings by right-clicking on the folder, selecting Properties, and selecting the Security tab.
3. Add the user account and allow write access.

By default, file uploads are limited to 4MB. Any attempt to upload a file larger than 4MB will result in an error message. The error message is generated by ASP.NET before any of your code runs; therefore, you have no control over the message being displayed.

You can change the maximum file size that can be uploaded by changing the `maxRequestLength` attribute of the `httpRuntime` element in the *web.config* file, as shown here (the value must be set in kilobytes):

```
<httpRuntime  executionTimeout="90"
    maxRequestLength="4096"
   useFullyQualifiedRedirectUrl="false"
    minFreeThreads="8" minLocalRequestFreeThreads="4"
    requestLengthDiskThreshold="1024"
    appRequestQueueLimit="100"/>
```

See Also

Chapter 3 for validation examples

Example 18-4. File upload (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH18FileUploadVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH18FileUploadVB"
 Title="File Upload" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  File Upload (VB)
 </div>
 <table width="90%" align="center" border="0">
  <tr>
   <td align="center">
    <asp:FileUpload ID="fuUpload" runat="server" />
   </td>
  </tr>
  <tr>
   <td align="center">
    <br />
    <input id="btnUpload" runat="server"
      type="button"
      value="Upload"
    onserverclick="btnUpload_ServerClick" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 18-5. File upload code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System.IO

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH18FileUploadVB.aspx
 ''' </summary>
 Partial Class CH18FileUploadVB
  Inherits System.Web.UI.Page
  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the upload button click
  ''' event. It is responsible saving the file to the local filesystem.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnUpload_ServerClick(ByVal sender As Object, _
                    ByVal e As System.EventArgs)
   'make sure file was specified and was found
   If (fuUpload.HasFile) Then
    fuUpload.SaveAs(Server.MapPath("uploads") & _
     "\" & fuUpload.FileName)
   End If
  End Sub 'btnUpload_ServerClick
 End Class 'CH18FileUploadVB
End Namespace
```

Example 18-6. File upload code-behind (.cs)

```
using System;
using System.IO;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH18FileUploadCS.aspx
 /// </summary>
 public partial class CH18FileUploadCS : System.Web.UI.Page
 {
   ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the upload button click
  /// event. It is responsible saving the file to the local filesystem.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnUpload_ServerClick(Object sender,
         System.EventArgs e)
  {
   // make sure file was specified and was found
   if (fuUpload.HasFile)
   {
    fuUpload.SaveAs(Server.MapPath("uploads") +
     "\\" + fuUpload.FileName);
   }
  } // btnUpload_ServerClick
 } // CH18FileUploadCS
}
```

Recipe 18.4. Processing an Uploaded File Without Storing It on the Filesystem

Problem

You want a user to be able to upload a file to the web server for immediateprocessing without having to first store the file.

Solution

Implement the solution described in Recipe 18.2. Instead of writing the file to the filesystem, use the input stream containing the uploaded file to process the data.

For the *.aspx* file, follow the steps for implementing the*.aspx* file in Recipe 18.2 and then, if you like, add a control that will show the results of theprocessingfor example, a `GridView` control to display the contents of an uploaded XML file.

In the code-behind file for the page, use the .NET language of your choice to:

1. Verify, in the Upload button click event handler, that the file has been uploadedthat is, that the `HasFile` property of the `FileUpload` control is set to `TRue` and (if appropriate) that it is a valid XML file.

2. Load the updated data and, if you elected to include a control for showing the contents of the uploaded file, bind the uploaded data to the controlfor example, a`GridView` control.

Examples 18-8 , 18-9 through 18-10 show the *.aspx* file and VB and C# code-behind files for an application we've written to illustrate this solution. The initial output is identical to Recipe 18.2's example output and is shown in Figure 18-3.

Discussion

This recipe demonstrates the concept of uploading files and processing them without having to store them to the local filesystem, as you might do with the contents of an XML file for example. This eliminates the problems of files being uploaded with the same names, inadvertently filling the hard drive, and the security aspects of allowing the ASP.NET write privileges on the local filesystem.

The example we've written to illustrate this solution is similar to the one described in Recipe 18.2, except that instead of saving the file contents to the filesystem, we process it immediately by loading the uploaded data into a `DataSet` and binding the `DataSet` to a `GridView` . What's more, our example uses the basic *.aspx* file described in Recipe 18.2 with a few changes to support using the page for uploading an XML file and displaying its contents.

In the Page_Load method of the code-behind, the table containing the upload controls is made visible and the GridView invisible. (We'll switch the visibility of the two at a later stage to display the contents of the processed file.)

When the user clicks the Upload button, the btnUpload_ServerClick method is executed. The code for ensuring a file was uploaded is identical to our example code in Recipe 18.2, except that, because thi example is expecting XML data, an additional check is added to ensure the uploaded file is an XML file. Since the same page is used to display the contents of the uploaded file, the table containing the upload controls needs to be made invisible and the GridView used to display the contents of the uploaded file made visible.

Next, we use the File Content property of the FileUpload control to get the file content to load the data into the DataSet . In our example, the XML file shown in Example 18-7 is being uploaded and, because of its formatting, can be read directly into the DataSet . With other types of data, you may need to do other processing. In addition, production code should validate the content type of the posted file and the contents of the file to ensure the uploaded file is valid.

The last step in our example is to bind the GridView on the form to the DataSet containing the uploaded XML data. Figure 18-4 shows the output for the uploaded and processed file. For more information on data binding, refer to the recipes in Chapter 2

Figure 18-4. Uploaded and processed file output

See Also

Chapter 3 for validation controls; FileUpload class documentation in the MSDN library for more information on file uploads

Example 18-7. uploaded XML file

```
<Root>
 <Book>
   <BookID>1</BookID>
  <Title>Access Cookbook</Title>
   <ISBN>0-596-00084-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>2</BookID>
  <Title>ASP.NET Cookbook</Title>
   <ISBN>0-596-00378-1</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>3</BookID>
  <Title>Perl Cookbook</Title>
   <ISBN>1-565-92243-3</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>4</BookID>
  <Title>Java Cookbook</Title>
   <ISBN>0-596-00170-3</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>5</BookID>
  <Title>JavaScript Application Cookbook</Title>
   <ISBN>1-565-92577-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>6</BookID>
  <Title>VB .Net Language in a Nutshell</Title>
   <ISBN>0-596-00092-8</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>7</BookID>
  <Title>Programming Visual Basic .Net</Title>
   <ISBN>0-596-00093-6</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>8</BookID>
  <Title>Programming C#</Title>
   <ISBN>0-596-00117-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>9</BookID>
  <Title>.Net Framework Essentials</Title>
   <ISBN>0-596-00165-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>10</BookID>
  <Title>COM and .Net Component Services</Title>
   <ISBN>0-596-00103-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
</Root>
```

Example 18-8. Upload file and process (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH18FileUploadAndProcessVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH18FileUploadAndProcessVB"
 Title="File Upload And Process" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  File Upload And Process (VB)
 </div>
  <!-- The following table is displayed when the user is
    uploading a file -->
  <table id="tabUpload" runat="server"
    align="center"
     width="60%"
     border="0">
   <tr>
    <td align="center">
    <asp:FileUpload ID="fuUpload" runat="server" />
    </td>
   </tr>
   <tr>
    <td align="center">
    <br />
    <input id="btnUpload" runat="server"
    type="button"
    value="Upload"
    onserverclick="btnUpload_ServerClick" />
   </td>
  </tr>
 </table>

 <!-- The following gridView is displayed to show the data from
   the uploaded file -->
 <asp:GridView ID="gvUploadedData" runat="server"
     AutoGenerateColumns="true"
     BorderColor="#000080"
     BorderStyle="Solid"
     BorderWidth="2px"
     Caption=""
     HorizontalAlign="Center"
     Width="90%" >
  <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
  <RowStyle cssClass="tableCellNormal" />
  <AlternatingRowStyle cssClass="tableCellAlternating" />
 </asp:GridView>
</asp:Content>
```

Example 18-9. Upload file and process code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System.Data
Imports System.IO
Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH18FileUploadAndProcessVB.aspx
 ''' </summary>
 Partial Class CH18FileUploadAndProcessVB
  Inherits System.Web.UI.Page
   '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   If (Not Page.IsPostBack) Then
    'make the table containing the upload controls visible and
    'the gridview with the uploaded data invisible
    tabUpload.Visible = True
    gvUploadedData.Visible = False
   End If
  End Sub 'Page_Load

   '*********************************************************************
  '
  ' ROUTINE: btnUpload_ServerClick
  '
  ' DESCRIPTION:
  '-----------------------------------------------------------------------
  ''' <summary>
  ''' This routine provides the event handler for the upload button click
  ''' event. It is responsible saving the file to the local filesystem.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnUpload_ServerClick(ByVal sender As Object, _
             ByVal e As System.EventArgs)
   Dim dSet As DataSet

   'make sure file was specified and was found
   If ((fuUpload.HasFile) AndAlso _
     (fuUpload.PostedFile.ContentType.Equals("text/xml"))) Then
    'make the table containing the upload controls invisible and
    'the datagrid with the uploaded data visible
    tabUpload.Visible = False
    gvUploadedData.Visible = True

    'load uploaded data into the dataset
    dSet = New DataSet
    dSet.ReadXml(fuUpload.FileContent)
    'bind the data to the datagrid on the form
    gvUploadedData.DataSource = dSet
    gvUploadedData.DataBind()
   Else
    'production code should notify user of upload error here
   End If
  End Sub 'btnUpload_ServerClick
 End Class 'CH18FileUploadAndProcessVB
End Namespace
```

Example 18-10. Upload file and process code-behind (.cs)

```
using System;
using System.Data;
using System.IO;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH18FileUploadAndProcessCS.aspx
 /// </summary>
 public partial class CH18FileUploadAndProcessCS : System.Web.UI.Page
 {
  ///************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   if (!Page.IsPostBack)
   {
    // make the table containing the upload controls visible and
    // the gridview with the uploaded data invisible
    tabUpload.Visible = true;
    gvUploadedData.Visible = false;
   }
  } // Page_Load

  ///************************************************************************
  /// <summary>
  /// This routine provides the event handler for the upload button click
  /// event. It is responsible saving the file to the local filesystem.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnUpload_ServerClick(Object sender,
           System.EventArgs e)
  {
   DataSet dSet;

   // make sure file was specified and was found
   if ((fuUpload.HasFile) &&
    (fuUpload.PostedFile.ContentType.Equals("text/xml")))
   {
    // make the table containing the upload controls invisible and
    // the datagrid with the uploaded data visible
    tabUpload.Visible = false;
    gvUploadedData.Visible = true;

    // load uploaded data into the dataset
    dSet = new DataSet();
    dSet.ReadXml(fuUpload.FileContent);

    // bind the data to the datagrid on the form
    gvUploadedData.DataSource = dSet;
    gvUploadedData.DataBind();
```

```
    }
    else
    {

     // production code should notify user of upload error here
    }
   } // btnUpload_ServerClick
  } //  CH18FileUploadAndProcessCS
}
```

PREV

PREV

Recipe 18.5. Storing the Contents of an Uploaded File in a Database

Problem

You need to provide the ability for a user to upload a file to the web server that will be processed later, so you want to store the file in the database.

Solution

Implement the solution described in Recipe 18.2. When the user clicks a button to initiate the upload process, instead of writing the file to the filesystem, use the input stream containing the uploaded fil along with ADO.NET to write the file to a database.

For the *.aspx* file, follow the steps for implementing the *.aspx* file in Recipe 18.2.

In the code-behind class for the page, use the .NET language of your choice to:

1. Process the Upload button click event and verify that a file has been uploaded.
2. Open a connection to the database.
3. Build the command used to add the data to the database and insert the file data.

The application we've written to demonstrate this solution uses the same *.aspx* file as Recipe 18.2's example (see Example 18-4). The code-behind for our application is shown in Examples 18-11 (VB) and 18-12 (C#). The initial output is the same as Recipe 18.2's example output and is shown in Example 18-3.

Discussion

Storing an uploaded file in a database is useful when a complete, unmodified upload record is required to be set apart from the web server's filesystem, when the file contains sensitive information, or when additional metadata needs to be stored with the file. It is common to store the uploaded data in a database and then process the data immediately or by another program outside of the web application. We will not go into that here, though.

The example we've written to demonstrate this solution includes a button to initiate the upload process and uses the input stream containing the uploaded file along with ADO.NET to write the file t a database. The example uses the same code as Recipe 18.2, changing only the actions performed ir the btnUpload_ServerClick method of the code-behind. After verifying that a file has been uploaded, a connection is made to the database.

We then create an OleDbCommand with the CommandText property set to a parameterized SQL INSERT statement to store the filename, the file size, and thecontents of the file in the database. We use a parameterized query to handle the binary data contained in the file.

The FileData column of our database needs to be able to handle the binary data contained in the file For SQL Server, the data type should be VarBinary or image. Even if the uploaded files are text files, use a binary field for storage of the data. Text files can contain Unicode or utf-8-encoded characters that SQL Server cannot store in text fields, which results in a SQL exception being thrown.

Next, three parameters are added to the parameter collection of the command object and the values are set with the uploaded file information. Because our example usesOleDb, which does not support named parameters as the SQL provider does, the parameters must be added in the same order they appear in the INSERT statement.

The Filename and Filesize parameters require creating the parameter and setting the value. The Filedata parameter is created in the same manner; however, the value must be set to a byte array. A byte array of the uploaded file data is available using theFileBytes property of the FileUpload control.

The last step is to set the connection property of the command to the connection opened earlier and executing the command. The ExecuteNonQuery method of the command object is used because no data is being returned by the command.

See Also

Recipe 18.2 for the base code used for this recipe and a discussion of the size limits ouploaded files

Example 18-11. Storing uploaded file to database code-behind (.vb)

```
Protected Sub btnUpload_ServerClick(ByVal sender As Object, _
      ByVal e As System.EventArgs)
 Dim dbConn As OleDbConnection = Nothing
 Dim dcmd As OleDbCommand
 Dim strConnection As String

 Try
  'make sure file was uploaded
  If (fuUpload.HasFile) Then
   'get the connection string from web.config and open a connection
   'to the database
   strConnection = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
   dbConn = New OleDbConnection(strConnection)
   dbConn.Open()
   'build the command used to add the data to the database
   dcmd = New OleDbCommand
   dcmd.CommandText = "INSERT INTO FileUpload " & _
       "(Filename, Filesize, FileData) " & _
       "VALUES " & _
       "(?, ?, ?)"

   'create the paramters and set the values for the file data
   dcmd.Parameters.Add(New OleDbParameter("Filename", _
          fuUpload.FileName))

   dcmd.Parameters.Add(New OleDbParameter("Filesize", _
        fuUpload.PostedFile.ContentLength))

   dcmd.Parameters.Add(New OleDbParameter("FileData", _
          fuUpload.FileBytes))

   'insert the file data
   dcmd.Connection = dbConn
   dcmd.ExecuteNonQuery()
  End If
 Finally
  If (Not IsNothing(dbConn)) Then
   dbConn.Close()
  End If
 End Try
End Sub 'btnUpload_ServerClick
```

Example 18-12. Storing uploaded file to database code-behind (.cs)

```
protected void btnUpload_ServerClick(Object sender,
       System.EventArgs e)
{
 OleDbConnection dbConn = null;
 OleDbCommand dcmd = null;
 String strConnection;

 try
 {
  // make sure file was uploaded
  if (fuUpload.HasFile)
  {
   // get the connection string from web.config and open a connection
   // to the database
   strConnection = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
   dbConn = new OleDbConnection(strConnection);
```

```
    dbConn.Open();

    // build the command used to add the data to the database
    dcmd = new OleDbCommand();
    dcmd.CommandText = "INSERT INTO FileUpload " +
        "(Filename, Filesize, FileData) " +
        "VALUES " +
        "(?, ?, ?)";

    // create the paramters and set the values for the file data
    dcmd.Parameters.Add(new OleDbParameter("Filename",
             fuUpload.FileName));

    dcmd.Parameters.Add(new OleDbParameter("Filesize",
         fuUpload.PostedFile.ContentLength));

    dcmd.Parameters.Add(new OleDbParameter("FileData",
             fuUpload.FileBytes));
    // insert the file data
    dcmd.Connection = dbConn;
    dcmd.ExecuteNonQuery();
   }
  }
  finally
  {
   if (dbConn != null)
   {
    dbConn.Close();
   }
  }
} // btnUpload_ServerClick
```

PREV

Chapter 19. Performance

PREV

19.0 Introduction

Performance has been a concern of ours throughout this book, and we have endeavored to provide you with production-ready code that will perform well in any setting. And when a recipe involves trade-offs between performance and ease of implementation, we strive to bring these to your attention. Nevertheless, when an application is not performing as well as you would like, you can improve matters by altering its handling of the following elements:

`ViewState`

> You can often improve a page's performance by disabling the `ViewState` for the page or some of its controls, but you have to be aware of the consequences.

String concatenation

> You've probably heard that it is better to use the `StringBuilder` object to build strings rather than the classic concatenation operators (& and +). But you may be wondering how much better it is and if it applies to your situation.

Data access

> With the different options available for data access, ways to improve data access performance exist, especially when choosing between the two primary methods for reading data from a databasei.e., via a `DataReader` or a `DataAdapter`.

SQL Server managed provider

> For the sake of database interoperability, the bulk of the recipes in this book show how to access data using the `OleDB` managed provider. Yet because of the performance that can be garnered, much can be said for using the SQL Server managed provider instead when you know the application will always access SQL Server 7.0 or later.

All of these topics are addressed in the recipes in this chapter.

Like all other programming tools, ASP.NET and the common language runtime (CLR) provide many different ways to accomplish a given task. And because each application is unique and there is no on "right" way to approach it, every application's performance is worthy of review, mitigated by its frequency of use and its significance. With this in mind, you may want to consider these performance-oriented recipes as much for their approaches to performance tuning as for their line-for-line coding techniques.

As you evaluate the comparisons we've made in this chapter between different data access methods, you should know that the measurements were made on a 1.7 GHz Pentium 4 PC with 1GB of memory. Your mileage may vary.

> The side-by-side test results presented in this chapter's examples should be used to compare the relative differences between data access methods. The time to retrieve data is a function of the hardware, the database, the fragmentation of the data, and other variables.

PREV

PREV

Recipe 19.2. Reducing Page Size by Selectively Disabling the ViewState

Problem

You want to reduce the size of your application pages to improve performance.

Solution

Review each page of your application and each of its controls to determine if the `ViewState` is required. Disable the `ViewState` where it is not explicitly needed.

In the code for the page, use the .NET language of your choice to do either of the following:

- Disable the `ViewState` for the page by setting the `EnableViewState` attribute in the `@ Page` directive to `False`. (Alternatively, set `Page.EnableViewState` to `False` in the code-behind.)
- Disable the `ViewState` for individual controls on the page by setting the control's `EnableViewState` attribute to `False`. (Alternatively, set the control's `EnableViewState` property to `False` in the code-behind.)

To illustrate these performance improvements, we took two examples from Chapter 2 and optimized them by disabling the `ViewState`. In the first example, we took the ASP. NET page created for Recipe 2.22, which displays a grid containing books and price data, and disabled the `ViewState` at the page level. Table 19-1 shows the page and `ViewState` size before and after the optimization.

Table 19-1. ViewState performance improvement for Recipe 2.22 example

	Before optimization	After optimization
Page size	9,947 bytes	6,766 bytes
`ViewState size`	3,362 bytes	162 bytes

In the second example, we have used the ASP.NET page created in Recipe 2.10 and disabled the `ViewState` for the row controls within the `DataGrid` that appears within the page body. Example 19-1 shows the *.aspx* file for this application. The code-behind class for the application is shown in Example 19-2 (VB) and Example 19-3 (C#). Table 19-2 shows the page and `ViewState` sizes before and after optimization.

Table 19-2. ViewState performance improvement for Recipe 2.10 example

	Before optimization	After optimization
Page size	9,448 bytes	6,378 bytes
`ViewState size`	3,734 bytes	670 bytes

Discussion

The `ViewState` is used to keep track of the state of each control on a page and to rehydrate the control upon postback to the server. Because of its ability to maintain state when a page is posted back to the server, the use of the `ViewState` reduces the amount of code you would have to write. Thanks to the `ViewState`, you no longer need to extract values from the posted form for processing or reset the control values when you display the page again, as is the case with classic ASP. The controls are accessed as they were when the page was initially generated.

Though use of the `ViewState` significantly reduces your coding and maintenance efforts, it comes at a cost. All of the data required to keep track of the control's state is stored in a hidden input control in the HTML page as shown next. Depending on the number and types of controls you use on your pages, the `ViewState` can get large, resulting in a decrease in performance. Because the `ViewState` data is sent to the browser when the page is rendered and returned to the server as part of the postback, a performance hit occurs when the page is first displayed and when the page is posted back to the server. Performance is degraded not so much by the generation of the `ViewState` data when the page is first rendered, but rather by the transfer of the extra `ViewState` data to and from the browser on postbacks, as well as by the processing of the data by the browser. Here is a typical `ViewState` input control:

```
<input type="hidden" name="_ _VIEWSTATE"
     value="dDwtOTQzNjg3NDE1O3Q8O2w8aTwxPjs"/>
```

While "byte counters" will be quick to disable the `ViewState` completely because of its inevitable negative impact on performance, a compromise is available that provides the best of both worlds: selectively disable `ViewState` because it is not needed for all pages or controls. By reviewing each of the pages in your application, you can improve the application's performance without losing the benefits of the `ViewState`.

The first step when reviewing a page is to determine if the page does a postback to itself. If not, then the `ViewState` can be disabled for the entire page. This is done by setting the `EnableViewState` attribute in the `@ Page` directive to false:

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
```

```
    AutoEventWireup="false"
     CodeFile="CH19ViewStatePerformanceVB1.aspx.vb"
      Inherits="ASPNetCookbook.VBExamples.CH19ViewStatePerformanceVB1"
    Title="ViewState Performance"
    EnableViewState="false" %>
```

Alternately, you can place this line of code in the `Page_Load` method to disable the `ViewState` for the entire page:

VB

```
  Page.EnableViewState = False
```

C#

```
  Page.EnableViewState = False;
```

Even with the `ViewState` disabled for a page, a few bytes will remain in the `value` setting of the hidden input control. If you are determined to remove all traces of the `ViewState`, you must remove the `form` element or remove the `runat="server"` attribute from the `form` element. Either action can cause maintenance issues later, and the resulting savings of fewer than 50 bytes in a 20KB page has no measurable performance impact, so we do not recommend this remedy.

If the page does a postback to itself, you will need to review each of the controls on the page. For each control, you need to determine if any state information is required by the control upon postback. If no state information is required, the `ViewState` for the control can be disabled.

The example page created in Recipe 2.22 displays a grid containing books and price data. The page has no "action" controls; therefore, this page has no mechanism to postback to itself and is a good candidate for disabling the `ViewState` at the page level, a conclusion presented in the results shown ir Table 19-1. After this optimization, the page size is 68% of the original size and the `ViewState` represents less than 2.4% of the optimized page.

The example page created in Recipe 2.10 is a good candidate for performance improvement. This page is similar to the page created in Recipe 2.22 but has three "action" controls used to sort the dat in the grid. Clicking on the column headers in the grid causes the page to be posted back to itself with the data sorted by the column clicked; therefore, this page cannot have the `ViewState` disabled at the page level.

Because the `ViewState` cannot be disabled at the page level, we need to review each control to determine if the `ViewState` is needed. The page contains two controls, a content control and a `DataGrid` control. The content control contains no "action" controls and no programmatically set content; therefore, the `ViewState` for the content control can be disabled using the code shown here:

```
  <asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody"
     enableviewstate="false">
```

While you might be tempted to disable the ViewState for the DataGrid, because all of the data is regenerated on each postback, you cannot. ASP.NET needs the ViewState information for the controls within the header of the DataGrid to process its click events and to execute the dgBooks_SortCommand method. If you disable the ViewState for the DataGrid, the postback will occur but none of the event handlers will be called.

A DataGrid is a container of controls. At its highest level, a DataGrid consists of a header control and one or more row controls. In this example, only the header contains "action" controls and because the data in each row are regenerated with each postback, the ViewState for the row controls can be disabled using the code shown here:

VB

```
For Each item In dgBooks.Items
 item.EnableViewState = False
Next
```

C#

```
foreach (DataGridItem item in dgBooks.Items)
{
 item.EnableViewState = false;
}
```

Code that programmatically disables the ViewState of individual controls, must be executed every time the page is rendered. In addition, the disabling of controls within a DataGrid must be performed after data binding.

The results in Table 19-2 confirm the advantage of this optimization. By disabling the ViewState for the content control and for each row in the DataGrid, we have reduced the size of the ViewState and the overall page size as well. After optimization, the page size is 68% of the original size and the ViewState represents less than 11% of the optimized page.

ASP.NET 2.0 has removed the information needed for control postback from the ViewState; the postback information is now contained within the ControlState. This separation of control functionality from data provides more flexibility in disabling the ViewState to reduce the size of pages. The ControlState is implemented in most ASP.NET server controls. The DataGrid is one of the server controls that does not use the ControlState.

If a GridView had been used in this example instead, such as the one shown in Recipe 2.14, the ViewState for the entire GridView could have been disabled without affecting the functionality of the page.

See Also

Recipes 2.10, 2.14, and 2.22

Example 19-1. Modified .aspx file from Recipe 2.10

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH19ViewStatePerformanceVB2.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH19ViewStatePerformanceVB2"
 Title="View State Performance" %>
<asp:Content ID="pageBody" Runat="server" ContentPlaceHolderID="PageBody"
   enableviewstate="false">
 <div align="center" class="pageHeading">
  Improving ViewState Performance of Recipe 2-10 (VB)
 </div>
 <asp:DataGrid id="dgBooks" runat="server"
     BorderColor="#000080"
     BorderWidth="2px"
    HorizontalAlign="Center"
    AutoGenerateColumns="False"
    Width="90%"
    AllowSorting="True"
    OnSortCommand="dgBooks_SortCommand" >
  <HeaderStyle HorizontalAlign="Center" CssClass="tableHeader" />
  <ItemStyle cssClass="tableCellNormal" />
  <AlternatingItemStyle cssClass="tableCellAlternating" />

  <Columns>
   <asp:BoundColumn DataField="Title"
    SortExpression="Title" />
   <asp:BoundColumn DataField="ISBN"
    ItemStyle-HorizontalAlign="Center"
    SortExpression="ISBN" />
   <asp:BoundColumn DataField="Publisher"
    ItemStyle-HorizontalAlign="Center"
    SortExpression="Publisher" />
  </Columns>
 </asp:DataGrid>
</asp:Content>
```

Example 19-2. Optimized code-behind for Recipe 2.10 (.vb)

```
Option Explicit On
Option Strict On

Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
  ''' <summary>
  ''' This class provides the code-behind for

   '''  CH19ViewStatePerformanceVB2.aspx
  ''' </summary>
   Partial Class CH19ViewStatePerformanceVB2
 Inherits System.Web.UI.Page
 'the following enumeration is used to define the sort orders
 Private Enum enuSortOrder
   soAscending = 0
   soDescending = 1
 End Enum
 'strings to use for the sort expressions and column title
 'separate arrays are used to support the sort expression and titles
 'being different
 Private ReadOnly sortExpression() As String = {"Title", "ISBN", "Publisher"}
 Private ReadOnly columnTitle() As String = {"Title", "ISBN", "Publisher"}

 'the names of the variables placed in the viewstate
 Private Const VS_CURRENT_SORT_EXPRESSION As String = "currentSortExpression"
 Private Const VS_CURRENT_SORT_ORDER As String = "currentSortOrder"

 '''*********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
      Dim defaultSortExpression As String
   Dim defaultSortOrder As enuSortOrder
   If (Not Page.IsPostBack) Then
  'sort by title, ascending as the default
   defaultSortExpression  =  sortExpression(0)
  defaultSortOrder = enuSortOrder.soAscending

  'store current sort expression and order in the viewstate then
  'bind data to the DataGrid
  ViewState(VS_CURRENT_SORT_EXPRESSION) = defaultSortExpression

  ViewState(VS_CURRENT_SORT_ORDER) = defaultSortOrder
  bindData(defaultSortExpression, _
        defaultSortOrder)
  End If

  'disable the ViewState for controls that do not need it
  disableViewState()
 End Sub 'Page_Load

 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the datagrid sort event.
 ''' It is responsible rebinding the data to the datagrid by the selected
 ''' column.
 ''' </summary>
 '''
 ''' <param name="source">Set to the source of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Protected Sub dgBooks_SortCommand(ByVal source As Object, _
          ByVal e As DataGridSortCommandEventArgs)
      Dim newSortExpression As String
   Dim currentSortExpression As String
   Dim currentSortOrder As enuSortOrder

      'get the current sort expression and order from the viewstate
   currentSortExpression = CStr(ViewState(VS_CURRENT_SORT_EXPRESSION))
   currentSortOrder = CType(ViewState(VS_CURRENT_SORT_ORDER), enuSortOrder)

   'check to see if this is a new column or the sort order
   'of the current column needs to be changed.
   newSortExpression = e.SortExpression
   If (newSortExpression = currentSortExpression) Then
  'sort column is the same so change the sort order
  If (currentSortOrder = enuSortOrder.soAscending) Then
    currentSortOrder = enuSortOrder.soDescending
    Else
    currentSortOrder = enuSortOrder.soAscending
  End If
      Else
  'sort column is different so set the new column with ascending
  'sort order
  currentSortExpression = newSortExpression
  currentSortOrder = enuSortOrder.soAscending
   End If
   'update the view state with the new sort information
   ViewState(VS_CURRENT_SORT_EXPRESSION) = currentSortExpression
   ViewState(VS_CURRENT_SORT_ORDER) = currentSortOrder

   'rebind the data in the datagrid
   bindData(currentSortExpression, _
         currentSortOrder)
 End Sub 'dgBooks_SortCommand

 '''*************************************************************************
 ''' <summary>
 ''' This routine queries the database for the data to displayed and binds
 ''' it to the datagrid
 ''' </summary>
```

```
'''

''' <param name="sortExpression">Set to the sort expression to use for
'''      sorting the data</param>
''' <param name="sortOrder">Set to the requried sort order</param>
Private Sub bindData(ByVal sortExpression As String, _
      ByVal sortOrder As enuSortOrder)
   Dim dbConn As OleDbConnection = Nothing
  Dim da As OleDbDataAdapter = Nothing
  Dim dTable As DataTable = Nothing
  Dim strConnection As String
  Dim strSQL As String
  Dim index As Integer
  Dim col As DataGridColumn = Nothing
  Dim colImage As String
  Dim strSortOrder As String
  Try
 'get the connection string from web.config and open a connection
 'to the database
 strConnection = ConfigurationManager. _
   ConnectionStrings("dbConnectionString").ConnectionString
 dbConn = New OleDbConnection(strConnection)
 dbConn.Open( )

 'build the query string and get the data from the database
 If (sortOrder = enuSortOrder.soAscending) Then
   strSortOrder = " ASC"
 Else
   strSortOrder = " DESC"
 End If
 strSQL = "SELECT Title, ISBN, Publisher " & _
     "FROM Book " & _
    "ORDER BY " & sortExpression & _
    strSortOrder

      da = New OleDbDataAdapter(strSQL, dbConn)
 dTable = New DataTable
 da.Fill(dTable)

 'loop through the columns in the datagrid updating the heading to
 'mark which column is the sort column and the sort order
 For index = 0 To dgBooks.Columns.Count - 1
   col = dgBooks.Columns(index)

   'check to see if this is the sort column
   If (col.SortExpression = sortExpression) Then
  'this is the sort column so determine whether the ascending or
  'descending image needs to be included
  If (sortOrder = enuSortOrder.soAscending) Then
     colImage = " <img src='images/sort_ascending.gif' border='0'>"
  Else
     colImage = " <img src='images/sort_descending.gif' border='0'>"
```

```
             End If
         Else
     'This is not the sort column so include no image html
     colImage = ""
      End If 'If (col.SortExpression = sortExpression)

      'set the title for the column
      col.HeaderText = columnTitle(index) & colImage
       Next index

    'set the source of the data for the datagrid control and bind it
    dgBooks.DataSource = dTable
    dgBooks.DataBind()

     Finally
    'cleanup
    If (Not IsNothing(dbConn)) Then
      dbConn.Close()
    End If
     End Try
  End Sub 'bindData

  '''*************************************************************************
  ''' <summary>
  ''' This routine disables the ViewState for all controls on the page
  ''' that do not need to use it.
  ''' </summary>
   Private Sub disableViewState()
     Dim item As DataGridItem
     'disable the ViewState for each row in the DataGrid
     For Each item In dgBooks.Items
    item.EnableViewState = False
     Next item
  End Sub 'disableViewState
   End Class 'CH19ViewStatePerformanceVB2
End Namespace
```

Example 19-3. Optimized code-behind for Recipe 2.10 (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.Common;
using System.Data.OleDb;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
```

```
namespace ASPNetCookbook.CSExamples
{
  /// <summary>
  /// This class provides the code-behind for
  ///  CH19ViewStatePerformanceCS2.aspx

  /// </summary>
  public partial class CH19ViewStatePerformanceCS2 : System.Web.UI.Page
  {
 // the following enumeration is used to define the sort orders
 private enum enuSortOrder
 {
 soAscending = 0,
 soDescending = 1
    }

 // strings to use for the sort expressions and column title
 // separate arrays are used to support the sort expression and titles
 // being different
 static readonly String[] sortExpression =
     new String[] { "Title", "ISBN", "Publisher" };
 static readonly String[] columnTitle =
     new String[] { "Title", "ISBN", "Publisher" };

    // the names of the variables placed in the viewstate
 static readonly String VS_CURRENT_SORT_EXPRESSION = "currentSortExpression";;
 static readonly String VS_CURRENT_SORT_ORDER = "currentSortOrder";

 ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event. It
 /// is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 private void Page_Load(object sender, System.EventArgs e)
 {
   String defaultSortExpression;
   enuSortOrder defaultSortOrder;

   if (!Page.IsPostBack)
   {
     // sort by title, ascending as the default
   defaultSortExpression  =  sortExpression[0];
  defaultSortOrder = enuSortOrder.soAscending;

  // bind data to the DataGrid
  this.ViewState.Add(VS_CURRENT_SORT_EXPRESSION, defaultSortExpression);
  this.ViewState.Add(VS_CURRENT_SORT_ORDER, defaultSortOrder);
  bindData(defaultSortExpression,
        defaultSortOrder);
   }
   // disable the ViewState for controls that do not need it
   disableViewState();
 }  // Page_Load

 ///***********************************************************************
 /// <summary>
 /// This routine provides the event handler for the datagrid sort event.
 /// It is responsible rebinding the data to the datagrid by the selected
 /// column.
 /// </summary>
 ///
 /// <param name="source">Set to the source of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void dgBooks_SortCommand(Object source,
    System.Web.UI.WebControls.DataGridSortCommandEventArgs e)
 {
   String newSortExpression = null;
   String currentSortExpression = null;
   enuSortOrder currentSortOrder;

   // get the current sort expression and order from the viewstate
   currentSortExpression =
    (String)(this.ViewState[VS_CURRENT_SORT_EXPRESSION]);
   currentSortOrder =
    (enuSortOrder)(this.ViewState[VS_CURRENT_SORT_ORDER]);
   // check to see if this is a new column or the sort order
   // of the current column needs to be changed.
   newSortExpression = e.SortExpression;
   if (newSortExpression == currentSortExpression)
   {
     // sort column is the same so change the sort order
  if (currentSortOrder == enuSortOrder.soAscending)
  {
    currentSortOrder = enuSortOrder.soDescending;
  }
  else
  {
    currentSortOrder = enuSortOrder.soAscending;

  }
   }
   else
   {
     // sort column is different so set the new column with ascending
     // sort order
     currentSortExpression = newSortExpression;
     currentSortOrder = enuSortOrder.soAscending;
   }

   // update the view state with the new sort information
   this.ViewState.Add(VS_CURRENT_SORT_EXPRESSION, currentSortExpression);
   this.ViewState.Add(VS_CURRENT_SORT_ORDER, currentSortOrder);

   // rebind the data in the datagrid
   bindData(currentSortExpression,
      currentSortOrder);
  } // dgBooks_SortCommand


 ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page index changed
 /// event of the datagrid. It is responsible for setting the page index
 /// from the passed arguments and rebinding the data.
 /// </summary>
 ///
 /// <param name="source">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void dgBooks_PageIndexChanged(Object source,
     System.Web.UI.WebControls.DataGridPageChangedEventArgs e)
 {
   String currentSortExpression;
   enuSortOrder currentSortOrder;

   // set new page index and rebind the data
   dgBooks.CurrentPageIndex = e.NewPageIndex;

   // get the current sort expression and order from the viewstate
   currentSortExpression =
    (String)(this.ViewState[VS_CURRENT_SORT_EXPRESSION]);
   currentSortOrder =
    (enuSortOrder)(this.ViewState[VS_CURRENT_SORT_ORDER]);
   // rebind the data in the datagrid
   bindData(currentSortExpression,
      currentSortOrder);
 } // dgCustomers_PageIndexChanged

 ///*********************************************************************
 /// <summary>
 /// This routine queries the database for the data to displayed and binds
 /// it to the datagrid
 /// </summary>
 ///
 /// <param name="sortExpression">Set to the sort expression to use for
 ///       sorting the data</param>
 /// <param name="sortOrder">Set to the requried sort order</param>
 private void bindData(String sortExpression,
      enuSortOrder sortOrder)
  {
   OleDbConnection dbConn = null;
   OleDbDataAdapter da = null;
   DataTable dTable = null;
   String strConnection = null;
```

```
    String strSQL = null;
    int index = 0;
    DataGridColumn col = null;
    String colImage = null;
    String strSortOrder = null;

  try
  {
    // get the connection string from web.config and open a connection
    // to the database
    strConnection = ConfigurationManager.
        ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(strConnection);
       dbConn.Open();

      // build the query string and get the data from the database
   if (sortOrder == enuSortOrder.soAscending)
      {
     strSortOrder = " ASC";
    }
   else
      {
     strSortOrder = " DESC";
     }

   strSQL = "SELECT Title, ISBN, Publisher " +
      "FROM Book " +
      "ORDER BY " + sortExpression +
      strSortOrder;

   da = new OleDbDataAdapter(strSQL, dbConn);
   dTable = new DataTable();
   da.Fill(dTable);

   // loop through the columns in the datagrid updating the heading to
   // mark which column is the sort column and the sort order
   for (index = 0; index < dgBooks.Columns.Count; index++)
   {
     col = dgBooks.Columns[index];
     // check to see if this is the sort column
     if (col.SortExpression == sortExpression)
     {
    // this is the sort column so determine whether the ascending or
    // descending image needs to be included
    if (sortOrder == enuSortOrder.soAscending)
    {
       colImage = " <img src='images/sort_ascending.gif' border='0'>";
    }
    else
    {
       colImage = " <img src='images/sort_descending.gif' border='0'>";
    }
     }
     else
     {
         // This is not the sort column so include no image html
    colImage = "";
     } // if (col.SortExpression == sortExpression)

   // set the title for the column
   col.HeaderText = columnTitle[index] + colImage;
     }  // for index

     // set the source of the data for the datagrid control and bind it
     dgBooks.DataSource = dTable;
     dgBooks.DataBind();
    } // try

       finally
     {
   //clean up
   if (dbConn != null)
   {

     dbConn.Close();
   }
     } // finally
    } // bindData

    ///**********************************************************************
    /// <summary>
    /// This routine disables the ViewState for all controls on the page
    /// that do not need to use it.
    /// </summary>
    private void disableViewState()
    {
   // disable the ViewState for each row in the DataGrid
   foreach (DataGridItem item in dgBooks.Items)
   {
     item.EnableViewState = false;
   }
    } // disableViewState
  } //  CH19ViewStatePerformanceCS2
 }
```

PREV

Recipe 19.3. Speeding Up String Concatenation with a StringBuilder

Problem

You want to reduce the time spent concatenating strings in an application that performs this operation repeatedly.

Solution

Concatenate strings with a `StringBuilder` object instead of the classic `&` and `+` concatenation operators.

Examples 19-4, 19-5 through 19-6 show the *.aspx* file and the VB and C# code-behind files for our application that demonstrates the performance difference between using the classic string operators and a `StringBuilder` object to perform concatenation. Our example concatenates two strings repeatedly and calculates the average time per concatenation for the two approaches. The output of the application is shown in Figure 19-1.

Figure 19-1. Measuring string concatenation performance output

Discussion

In the CLR, strings are immutable, which means that once they have been created they cannot be changed. If you concatenate the two strings, `str1` and `str2`, shown in the following code fragment, the resulting value of `str1` will be `1234567890`:

VB

```
 str1 = "12345"
 str2 = "67890"
 str1 = str1 & str2
```

C#

```
 str1 = "12345";
 str2 = "67890";
 str1 = str1 + str2;
```

The way in which this concatenation is accomplished may come as a bit of a surprise to you. Since `str1` cannot be changed (it is immutable), it is disposed of and a new string `str1` is created that contains the concatenation of `str1` and `str2`. As you might expect, a lot of overhead is associated with this operation.

The `StringBuilder` object provides a faster method of concatenating strings. A `StringBuilder` object treats strings as an array of characters that can be altered without re-creating the object. When a `StringBuilder` object is created, the CLR allocates a block of memory in which to store the string. As characters are added to a `StringBuilder` object, they are stored in the available memory block. If the additional characters will not fit within the current block, additional memory is allocated to store the new data.

The default capacity of a `StringBuilder` is 16 characters, but the number can be set to any value up to 2,147,483,647 characters, the maximum size of an integer type. If you know approximately how long the final string will be, you can improve performance by setting the maximum size when the `StringBuilder` is created, which reduces the number of additional memory allocations that must be performed.

As Figure 19-1 shows, the performance difference between classic concatenation and concatenation using a `StringBuilder` object is dramatic. Classic concatenation averaged 2.6703 milliseconds per concatenation, while using a `StringBuilder` object averaged 0.0004 milliseconds per concatenation, which is nearly 7,000 times faster.

`StringBuilder` objects are not limited to concatenation: they can support inserting, removing, replacing, and appending formatted strings. The `StringBuilder` is a significant improvement over the classic manipulation of strings.

The question arises as to when classic string manipulation or a `StringBuilder` should be used. Every application is different. However, if you are performing a concatenation of fewer than 510 strings outside of a loop (done only once), you should probably use classic string manipulation because of the overhead of creating a `StringBuilder` object. Any time you are performing string manipulations within a loop or are combining many string fragments, a `StringBuilder` should be used. If you are unsure, create a test similar to the one we use in our examples for this recipe and measure the two approaches yourself.

See Also

In the MSDN Library, search for "Use StringBuilder for Complex String Manipulation."

Example 19-4. Measuring string concatenation performance (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
  AutoEventWireup="false"
   CodeFile="CH19StringConcatenationPerformanceVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH19StringConcatenationPerformanceVB"
  Title="String Concatenation Performance" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
  <div align="center" class="pageHeading">
    Page Heading (VB)
  </div>
  <table width="70%" align="center" border="0">
 <tr>
   <td>&nbsp;</td>
   <td align="center" class="subHeading">Time Per Concatenation (mSec)</td>
 </tr>
 <tr>
   <td class="labelText">Classic Concatentation</td>
   <td id="cellClassic" runat="server" align="center" class="labelText"></td>
 </tr>
 <tr>
   <td class="labelText">Using StringBuilder</td>
      <td id="cellSB" runat="server" align="center" class="labelText"></td>
 </tr>
  </table>
</asp:Content>
```

Example 19-5. Measuring string concatenation performance code-behind (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH19StringConcatenationPerformanceVB.aspx
 ''' </summary>
 Partial  Class CH19StringConcatenationPerformanceVB
  Inherits System.Web.UI.Page
  '''**********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
   Const STRING_SECTION As String = "1234567890"

   Dim testStr As String
   Dim testStrBuilder As StringBuilder
   Dim counter As Integer
   Dim startTime As DateTime
   Dim elapsedTime As TimeSpan
   Dim loops As Integer

   'measure the elapsed time for 10000 classic string concatenation
   loops = 10000
   startTime = DateTime.Now()
   testStr = ""
   For counter = 1 To loops
    testStr &= STRING_SECTION
   Next

   elapsedTime = DateTime.Now.Subtract(startTime)

   'set the table cell value to the average time per concatenation
   'in milliseconds
   cellClassic.InnerText = _
    (elapsedTime.TotalMilliseconds / loops).ToString("0.0000")

   'measure the elapsed time for 1,000,000  stringbuilder concatenations
   'NOTE: Many more loops were used to provide a measureable time period
   loops = 1000000
   startTime = DateTime.Now()
   testStrBuilder = New StringBuilder
   For counter = 1 To loops
    testStrBuilder.Append(STRING_SECTION)
   Next

   elapsedTime = DateTime.Now.Subtract(startTime)

   'set the table cell value to the average time per concatenation
   'in milliseconds
   cellSB.InnerText = _

    (elapsedTime.TotalMilliseconds / loops).ToString("0.0000")
  End Sub 'Page_Load
 End Class  'CH19StringConcatenationPerformanceVB
End Namespace
```

Example 19-6. Measuring string concatenation performance code-behind (.cs)

```
using System;
using System.Text;

namespace ASPNetCookbook.CSExamples

{
 /// <summary<
 /// This class provides the code-behind for
 ///  CH19StringConcatenationPerformanceCS.aspx
 /// </summary>
 public partial class CH19StringConcatenationPerformanceCS : \
  System.Web.UI.Page
 {
 ///***********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param<
 protected void Page_Load(object sender, EventArgs e)
 {
  const string STRING_SECTION = "1234567890";

  string testStr = null;
  StringBuilder testStrBuilder = null;
  DateTime startTime;
  TimeSpan elapsedTime;
  int counter;
  int loops;

  // measure the elapsed time for 10000  stringbuilder concatenations
  loops = 10000;
  startTime = DateTime.Now;
  testStr = "";
  for (counter = 1; counter <= loops; counter++)
  {
   testStr += STRING_SECTION;
  }

   elapsedTime = DateTime.Now.Subtract(startTime);
```

```
    // set the table cell value to the average time per concatenation
    // in milliseconds
    cellClassic.InnerText =
     (elapsedTime.TotalMilliseconds / loops).ToString("0.0000");

     // measure the elapsed time for 1,000,000 classic string concatenation
     // NOTE: Many more loops were used to provide a measureable time period
     loops = 1000000;
     startTime = DateTime.Now;
     testStrBuilder = new StringBuilder();
     for (counter = 1; counter <= loops; counter++)
     {
     testStrBuilder.Append(STRING_SECTION);
     }

     elapsedTime = DateTime.Now.Subtract(startTime);

     // set the table cell value to the average time per concatenation
     // in milliseconds
     cellSB.InnerText =
     (elapsedTime.TotalMilliseconds / loops).ToString("0.0000");
    } // Page_Load } // CH19StringConcatenationPerformanceCS }
```

PREV

Recipe 19.4. Speeding Up Read-Only Data Access

Problem

You want to speed up read-only data access to a database in your application.

Solution

Use a `DataReader` instead of a `DataAdapter` to access the data.

Examples 19-7, 19-8 through 19-9 show the *.aspx* file and VB and C# code-behind files for our application that demonstrates the performance difference between a `DataReader` and a `DataAdapter` using the `OleDB` managed provider. Figure 19-2 shows the output of the application. Refer to Recipe 19.4 for an equivalent example using the SQL Server managed provider.

Discussion

The CLR provides two primary methods for reading data from a database. The first is to use a `DataReader`, and the second is to use a `DataAdapter` in conjunction with a `DataTable` or `DataSet`.

Figure 19-2. Measuring data reader and data adapter performance output

The `DataReader` provides forward, read-only access to the data read from the database. It provides no mechanisms for randomly accessing the data.

A `DataAdapter`, along with a `DataTable` or `DataSet`, provides random access to data. In addition, the data can be changed in the `DataTable` or `DataSet`, and the `DataAdapter` can be used to update the data in the database.

Of the two access methods, the `DataReader` is the lightest and fastest and is preferable when you need to only read the data, as reflected in the results we show for our sample application inFigure 19-2. Our example reads 10KB and 100KB records from a SQL Server database table containing 500KB rows. The table contains five columns, of which three are retrieved in the query. The data indicates that using a `DataAdapter` is anywhere from 3868% slower than using a`DataReader`.

See Also

Recipe 19.4

Example 19-7. Measuring data reader and data adapter performance (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH19DataAccessPerformanceVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH19DataAccessPerformanceVB"
 Title="Data Access Performance" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Data Access Performance Using OleDB Provider (VB)
 </div>
 <table width="70%" align="center" border="0" class="dataEntry">
  <tr>
   <td align="center">Rows Read</td>
   <td align="center">OleDBDataReader Time (mSec)</td>
   <td align="center">OleDBDataAdaptor Time (mSec)</td>
  </tr>
  <tr>
   <td align="center">10,000</td>
   <td id="cellDR10K" runat="server" align="center"></td>
   <td id="cellDA10K" runat="server" align="center"></td>

  </tr>
  <tr>
   <td align="center">100,000</td>
   <td id="cellDR100K" runat="server" align="center"></td>
   <td id="cellDA100K" runat="server" align="center"></td>
  </tr>
 </table>
</asp:Content>
```

Example 19-8. Measuring data reader and data adapter performance code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 ''' CH19DataAccessPerformanceVB.aspx
 ''' </summary>
 Partial Class CH19DataAccessPerformanceVB
  Inherits System.Web.UI.Page

  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name='sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
   Dim dbConn As OleDbConnection = Nothing
   Dim strConnection As String
   Dim elapsedTime As TimeSpan

   Try
    'get the connection string from web.config and open connection
    'to the database
    strConnection = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDbConnection(strConnection)
    dbConn.Open()

    'get times for 10,000 records
    elapsedTime = getDataAdapterTime(dbConn, 10000)
    cellDA10K.InnerText = elapsedTime.TotalMilliseconds.ToString("0.0000")

    elapsedTime = getDataReaderTime(dbConn, 10000)
    cellDR10K.InnerText = elapsedTime.TotalMilliseconds.ToString("0.0000")

    'get times for 100,000 records
    elapsedTime = getDataAdapterTime(dbConn, 100000)
    cellDA100K.InnerText = elapsedTime.TotalMilliseconds.ToString("0.0000";)
    elapsedTime = getDataReaderTime(dbConn, 100000)
    cellDR100K.InnerText = elapsedTime.TotalMilliseconds.ToString("0.0000")

   Finally
    'clean up
    If (Not IsNothing(dbConn)) Then
    dbConn.Close()
    End If
   End Try
  End Sub 'Page_Load

  '''**************************************************************************
  ''' <summary>
  ''' This routine retrieves the passed number of records from the database
  ''' using an OleDBDataReader and returns the elapsed time
  ''' </summary>
  '''
  ''' <param name="dbConn">
  ''' Set to an open connection to the database
  ''' </param>
  ''' <param name="numberOfRecords">
  ''' Set to the number of records to read
  ''' </param>
  '''
  ''' <returns>
  ''' Timespan set to the elapsed time requried to read the data
  ''' </returns>
  Private Function getDataReaderTime(ByVal dbConn As OleDbConnection, _
          ByVal numberOfRecords As Integer) _
      As TimeSpan
   Dim dCmd As OleDbCommand = Nothing
   Dim dr As OleDbDataReader = Nothing
   Dim strSQL As String
   Dim startTime As DateTime
   Dim elapsedTime As TimeSpan
   Dim bookTitle As String
   Dim isbn As String
   Dim price As Decimal

   Try
    startTime = DateTime.Now()

    'build the query string and get the data from the database
    strSQL = "SELECT Top " & numberOfRecords.ToString() & " " & _
    "BookTitle, ISBN, Price " & _
    "FROM PerformanceTesting " & _
    "ORDER BY PerformanceTestingID"


    'read the data from the database
    dCmd = New OleDbCommand(strSQL, dbConn)
    dr = dCmd.ExecuteReader()
    Do While (dr.Read())
    bookTitle = CStr(dr.Item("BookTitle"))
    isbn = CStr(dr.Item("ISBN"))
    price = CDec(dr.Item("Price"))
    Loop

    'return the elapsed time
    elapsedTime = DateTime.Now.Subtract(startTime)
    getDataReaderTime = elapsedTime

   Finally
    If (Not IsNothing(dr)) Then
    dr.Close()
    End If
   End Try
  End Function 'getDataReaderTime

  '''*************************************************************************
  ''' <summary>
  ''' This routine retrieves the passed number of records from the database
  ''' using an OleDbDataAdapter and returns the elapsed time
  ''' </summary>
  '''
  ''' <param name="dbConn">
  ''' Set to an open connection to the database
  ''' </param>
  ''' <param name="numberOfRecords">
  ''' Set to the number of records to read
  ''' </param>
  '''
  ''' <returns>
  ''' Timespan set to the elapsed time requried to read the data
  ''' </returns>
  Private Function getDataAdapterTime(ByVal dbConn As OleDbConnection, _
        ByVal numberOfRecords As Integer) _
    As TimeSpan
   Dim da As OleDbDataAdapter
   Dim dTable As DataTable
   Dim strSQL As String
   Dim startTime As DateTime
   Dim elapsedTime As TimeSpan

   startTime = DateTime.Now()

   'build the query string and get the data from the database
   strSQL = "SELECT Top " & numberOfRecords.ToString() & " " & _
    "BookTitle, ISBN, Price " & _
    "FROM PerformanceTesting " & _
    "ORDER BY PerformanceTestingID"
   'read the data from the database
   da = New OleDbDataAdapter(strSQL, dbConn)
   dTable = New DataTable
   da.Fill(dTable)

   'return the elapsed time
   elapsedTime = DateTime.Now.Subtract(startTime)
   getDataAdapterTime = elapsedTime
  End Function 'getDataAdapterTime
 End Class 'CH19DataAccessPerformanceVB
End Namespace
```

Example 19-9. Measuring data reader and data adapter performance code-behind (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH19DataAccessPerformanceCS.aspx
 /// </summary>
 public partial class CH19DataAccessPerformanceCS : System.Web.UI.Page
 {

  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   OleDbConnection dbConn = null;
   String strConnection;
   TimeSpan elapsedTime;
   try
   {
    // get the connection string from web.config and open a connection
    // to the database
    strConnection = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(strConnection);
    dbConn.Open( );

    // get times for 10,000 records
    elapsedTime = getDataAdapterTime(dbConn, 10000);
    cellDA10K.InnerText = elapsedTime.TotalMilliseconds.ToString("0.0000");

    elapsedTime = getDataReaderTime(dbConn, 10000);
    cellDR10K.InnerText = elapsedTime.TotalMilliseconds.ToString("0.0000");

    // get times for 100,000 records
    elapsedTime = getDataAdapterTime(dbConn, 100000);
    cellDA100K.InnerText = elapsedTime.TotalMilliseconds.ToString("0.0000");

    elapsedTime = getDataReaderTime(dbConn, 100000);
    cellDR100K.InnerText = elapsedTime.TotalMilliseconds.ToString("0.0000");
   }

   finally
   {
    if (dbConn != null)
    {
    dbConn.Close();
    }
   }
  } // Page_Load

  ///***************************************************************************
  /// <summary>
  /// This routine retrieves the passed number of records from the database
  /// using an OleDBDataReader and returns the elapsed time
  /// </summary>
  ///
  /// <param name="dbConn">
  /// Set to an open connection to the database
  /// </param>
  /// <param name="numberOfRecords">
  /// Set to the number of records to read
  /// </param>
  ///
  /// <returns>
  /// Timespan set to the elapsed time requried to read the data
  /// </returns>

  private TimeSpan getDataReaderTime(OleDbConnection dbConn,
          int numberOfRecords)
  {
   OleDbCommand dCmd = null;
   OleDbDataReader dr = null;
   string strSQL = null;
   DateTime startTime;
```

```
 TimeSpan elapsedTime;
 String bookTitle;
 String isbn;
 Decimal price;

 try
 {
  startTime = DateTime.Now;

  // build the query string used to get the data from the database
  strSQL = "SELECT Top " + numberOfRecords.ToString() + " " +
  "BookTitle, ISBN, Price " +
  "FROM PerformanceTesting " +
  "ORDER BY PerformanceTestingID";

  // read the data from the database
  dCmd = new OleDbCommand(strSQL, dbConn);
  dr = dCmd.ExecuteReader();
  while (dr.Read())
  {
  bookTitle = (String)(dr["BookTitle"]);
  isbn = (String)(dr["ISBN"]);
  price = Convert.ToDecimal(dr["Price"]);
  }

  //return the elapsed time
  elapsedTime = DateTime.Now.Subtract(startTime);
  return (elapsedTime);

 }

 finally
 {
  // clean up
  if (dr != null)
  {

  dr.Close();
  }
 }
} // getDataReaderTime

///***********************************************************************
/// <summary>
/// This routine retrieves the passed number of records from the database
/// using an OleDbDataAdapter and returns the elapsed time
/// </summary>
///
/// <param name="dbConn">
/// Set to an open connection to the database
/// </param>
/// <param name="numberOfRecords">
```

```
  /// Set to the number of records to read
  /// </param>
  ///
  /// <returns>
  /// Timespan set to the elapsed time requried to read the data
  /// </returns>

  private TimeSpan getDataAdapterTime(OleDbConnection dbConn,
        int numberOfRecords)
  {
   OleDbDataAdapter da = null;
   DataTable dTable = null;

   string strSQL = null;
   DateTime startTime;
   TimeSpan elapsedTime;

   startTime = DateTime.Now;

   // build the query string used to get the data from the database
   strSQL = "SELECT Top " + numberOfRecords.ToString() + " " +
    "BookTitle, ISBN, Price " +
    "FROM PerformanceTesting " +
    "ORDER BY PerformanceTestingID";

   // read the data from the database
   da = new OleDbDataAdapter(strSQL, dbConn);
   dTable = new DataTable();
   da.Fill(dTable);

   // return the elapsed time
   elapsedTime = DateTime.Now.Subtract(startTime);
   return (elapsedTime);
  } // getDataAdapterTime
 } // CH19DataAccessPerformanceCS
}
```

Recipe 19.5. Speeding Up Data Access to a SQL Server Database Using the SQL Provider

Problem

You want to speed up data access in an application that will always be used with SQL Server.

Solution

Use the SQL Server managed provider instead of the `OleDB` managed provider for accessing the data in the database.

In the code-behind class for the page, open a connection to a SQL Server database using the `SQLConnection` class and then use the SqlCommand, SqlDataReader, and SqlDataAdapter objects as required by your application.

To test the SQL provider, we have implemented our example from Recipe 19.3 and replaced the `geTDataReaderTime` and `geTDataAdapterTime` methods in the code-behind with the code shown in Examples 19-10 (VB) and 19-11 (C#). The output of the test is shown in Figure 19-3.

Figure 19-3. Performance using SQL managed provider output

Discussion

The CLR provides four managed providers for accessing data in a database: SQL, `OleDB`, ODBC, and Oracle. The `OleDB` and ODBC providers can be used to access virtually any databaseincluding SQL Server, Access, Oracle, and many othersusing an `OleDB` (or ODBC) layer. `OleDB` communicates to a data source through the `OleDB` service component, which provides connection pooling and transaction services, and the `OleDB` provider for the data source. In contrast, the SQL Server provider uses a proprietary protocol to access SQL Server directly, eliminating the additional layer of the`OleDB` service component and thereby improving performance. It can only be used to access SQL Server 7.0 or a later release, however.

Comparing the results in Examples 19-10 (VB) and 19-11 (C#) with the results shown in Recipe 19.3 indicates that the SQL Server provider is faster than the `OleDB` provider when accessing SQL Server; access using a `DataReader` is approximately twice as fast, while access using a`DataAdapter` is 75100% faster.

The data also indicates that when using the SQL Server provider, the difference between using the `DataReader` and the `DataAdapter` is more significant. The `DataAdapter` is 8088% slower than the `DataReader` with the SQL Server provider.

See Also

Recipe 19.3; search for "The .NET Framework Data Provider for SQL Server" in the MSDN Library

Example 19-10. Methods using SQL provider (.vb)

```
'''*************************************************************************
''' <summary>
''' This routine retrieves the passed number of records from the database
''' using a SqlDataReader and returns the elapsed time
''' </summary>
'''
''' <param name="dbConn">
''' Set to an open connection to the database
''' </param>
''' <param name="numberOfRecords">
''' Set to the number of records to read

''' </param>
'''
''' <returns>
''' Timespan set to the elapsed time requried to read the data
''' </returns>
Private Function getDataReaderTime(ByVal dbConn As SqlConnection, _
      ByVal numberOfRecords As Integer) _ As TimeSpan
 Dim dCmd As SqlCommand = Nothing
 Dim dr As SqlDataReader = Nothing
 Dim strSQL As String
 Dim startTime As DateTime
 Dim elapsedTime As TimeSpan
 Dim bookTitle As String
 Dim isbn As String
 Dim price As Decimal

 Try
```

```
  startTime = DateTime.Now()

  'build the query string and get the data from the database
  strSQL = "SELECT Top " & numberOfRecords.ToString() & " " & _
    "BookTitle, ISBN, Price " & _
    "FROM PerformanceTesting " & _
    "ORDER BY PerformanceTestingID"

  'read the data from the database
  dCmd = New SqlCommand(strSQL, dbConn)
  dr = dCmd.ExecuteReader()
  Do While (dr.Read())
   bookTitle = CStr(dr.Item("BookTitle"))
   isbn = CStr(dr.Item("ISBN"))
   price = CDec(dr.Item("Price"))
  Loop

  'return the elapsed time
  elapsedTime = DateTime.Now.Subtract(startTime)
  getDataReaderTime = elapsedTime

 Finally
  If (Not IsNothing(dr)) Then
   dr.Close()
  End If
 End Try End Function 'getDataReaderTime

 '''*************************************************************************
 ''' <summary>
 ''' This routine retrieves the passed number of records from the database
 ''' using a SqlDataAdapter and returns the elapsed time
 ''' </summary>
 '''

''' <param name="dbConn">
''' Set to an open connection to the database
''' </param>
''' <param name="numberOfRecords">
''' Set to the number of records to read
''' </param>
'''
''' <returns>
''' Timespan set to the elapsed time requried to read the data
''' </returns>
Private Function getDataAdapterTime(ByVal dbConn As SqlConnection, _
      ByVal numberOfRecords As Integer) _
    As TimeSpan
 Dim da As SqlDataAdapter
 Dim dTable As DataTable
 Dim strSQL As String
 Dim startTime As DateTime
 Dim elapsedTime As TimeSpan
 startTime = DateTime.Now()

 'build the query string and get the data from the database
 strSQL = "SELECT Top " & numberOfRecords.ToString() & " " & _
   "BookTitle, ISBN, Price " & _
   "FROM PerformanceTesting " & _
   "ORDER BY PerformanceTestingID"

 'read the data from the database
 da = New SqlDataAdapter(strSQL, dbConn)
 dTable = New DataTable
 da.Fill(dTable)

 'return the elapsed time
 elapsedTime = DateTime.Now.Subtract(startTime)
 getDataAdapterTime = elapsedTime
End Function 'getDataAdapterTime
```

Example 19-11. Methods using SQL provider (.cs)

```
///**************************************************************************
/// <summary>
/// This routine retrieves the passed number of records from the database
/// using a SqlDataReader and returns the elapsed time
/// </summary>
///
/// <param name="dbConn">
/// Set to an open connection to the database
/// </param>
/// <param name="numberOfRecords">
/// Set to the number of records to read
/// </param>
///

/// <returns>
/// Timespan set to the elapsed time requried to read the data
/// </returns>
private TimeSpan getDataReaderTime(SqlConnection dbConn,
      int numberOfRecords)
{
 SqlCommand dCmd = null;
 SqlDataReader dr = null;
 string strSQL = null;
 DateTime startTime;
 TimeSpan elapsedTime;
 String bookTitle;
 String isbn;
```

```
  Decimal price;

  try
  {
   startTime = DateTime.Now;

   // build the query string used to get the data from the database
   strSQL = "SELECT Top " + numberOfRecords.ToString() + " " +
     "BookTitle, ISBN, Price " +
     "FROM PerformanceTesting " +
     "ORDER BY PerformanceTestingID";

   // read the data from the database
   dCmd = new SqlCommand(strSQL, dbConn);
   dr = dCmd.ExecuteReader();
   while (dr.Read())
   {
    bookTitle = (String)(dr["BookTitle"]);
    isbn = (String)(dr["ISBN"]);
    price = Convert.ToDecimal(dr["Price"]);

   }

   //return the elapsed time
   elapsedTime = DateTime.Now.Subtract(startTime);
   return (elapsedTime);

  }
  finally
  {
   // clean up
   if (dr != null)
   {
    dr.Close();
   }
  }
} // getDataReaderTime

///*********************************************************************
/// <summary>

/// This routine retrieves the passed number of records from the database
/// using a SqlDataAdapter and returns the elapsed time
/// </summary>
///
/// <param name="dbConn">
/// Set to an open connection to the database
/// </param>
/// <param name="numberOfRecords">
/// Set to the number of records to read
/// </param>
///
```

```
/// <returns>
/// Timespan set to the elapsed time requried to read the data
/// </returns>
private TimeSpan getDataAdapterTime(SqlConnection dbConn,
      int numberOfRecords)
{
 SqlDataAdapter da = null;
 DataTable dTable = null;
 string strSQL = null;
 DateTime startTime;
 TimeSpan elapsedTime;

 startTime = DateTime.Now;
 // build the query string used to get the data from the database
 strSQL = "SELECT Top " + numberOfRecords.ToString() + " " +
   "BookTitle, ISBN, Price " +
   "FROM PerformanceTesting " +
   "ORDER BY PerformanceTestingID";

 // read the data from the database
 da = new SqlDataAdapter(strSQL, dbConn);
 dTable = new DataTable();
 da.Fill(dTable);

 // return the elapsed time
 elapsedTime = DateTime.Now.Subtract(startTime);
 return (elapsedTime);
} // getDataAdapterTime
```

Chapter 20. HTTP Handlers

PREV

20.0 Introduction

An *HTTP handler* is a class that intercepts and handles requests for a resource of a given type on a web server. HTTP handlers are a key feature of ASP.NET. For instance, when you request an *.aspx* file, a built-in HTTP handler intercepts the request and takes charge of loading and executing the *.aspx* file. ASP.NET also provides built-in HTTP handlers for *.asmx*, *.ascx*, *.cs*, and *.vb* files, as well as other file types. The `<httpHandlers>` element of the *machine.config* file contains a list of the standard HTTP handlers configured for your web server.

Overriding ASP.NET's Built-in HTTP Handlers

The `<httpHandlers>` element in *machine.config* defines how ASP.NET handles requests for all of the standard file extensions found in most ASP.NET applications. These include *.aspx, .asmx, .ascx, .cs, .vb, .vbproj, .csproj, .soap*, and many others. By placing your own handler settings in *web.config*, you can override those defined in *machine.config*. The override maps incoming requests to the appropriate `IHttpHandler` class you define (see Recipe 20.1's "Discussion" section for more details). The override can be for a single URL or for all requests with a given extension.

You can extend the built-in handlers provided by ASP.NET or write your own. A custom HTTP handler is useful when you want to handle requests by your application for a given resource on your own. Fo example, custom handlers are useful for returning binary data, such as the contents of an image file, or for handling the processing necessary to access a resource stored in a database. HTTP handlers provide a good mechanism for building reusable assemblies for your web applications, such as a general purpose file download module able to handle requests for almost any file type. Each of these ideas is illustrated in the recipes in this chapter.

HTTP handlers are similar to the ISAPI extensions used to implement classic ASP for IIS. However, whereas ISAPI extensions are difficult to implement and can only be implemented in C++, HTTP handlers are supported by ASP.NET and can be implemented in any .NET language.

PREV

Recipe 20.2. Creating a Reusable Image Handler

Problem

You want to create a reusable assembly that retrieves image data from a database and processes it before sending it to a browser.

Solution

Create an HTTP handler to read the image data from the database and send it to the browser.

To implement a custom, reusable HTTP handler:

1. Create a separate Class Library project in Visual Studio.
2. Create a class in the project that implements the `IHttpHandler` interface and then place code to handle the request in the `ProcessRequest` method.
3. Compile the project as an assembly and place the assembly in the*bin* directory of your web project.
4. Add an `<httpHandlers>` element to the *web.config* file in your web project referencing your custom HTTP handler.
5. Reference the URL of the HTTP handler in your application.

Examples 20-3 and 20-4 show the VB and C# class files we've written to implement an image handle as an HTTP handler. Examples 20-5 , 20-6 through 20-7 show the *.aspx* file and VB and C# code-behind files for our application that demonstrates the use of the HTTP handler.

Discussion

HTTP handlers are classes that implement the `IHttpHandler` interface. Implementing the `IHttpHandler` interface requires the implementation of two methods: `IsReusable` and `ProcessRequest` . `IsReusable` is a property that explicitly returns a Boolean value that indicates if the HTTP handler can be reused by other HTTP requests. For synchronous handlers, like our example, the property should always return `false` so the handler is not pooled (kept in memory). The `ProcessRequest` method is where the work is performed and you should place code that processes the requests here.

To create a *reusable* HTTP handler, you need to eliminate all application-specific code from the class. You must compile the class as a separate .NET assembly and place the assembly in the*bin* directory of each application that uses it.

To create an assembly that contains only the handler code, you need to create a separate Class

Library project in Visual Studio. In our example, we have named the project`CH20 ImageHandlerVB` (or `CH20ImageHandlerCS` for C#), resulting in an assembly that has the same name as the project. We then compile the assembly, place it in the*bin* directory of the web project, and add a reference to the assembly.

> When you create a new Class Library project for your HTTP handler, you will need to add a reference to the `System.Web` assembly. This is required because the `IHttpHandler` interface and `HttpContext` class used by the HTTP handler are defined in the `System.Web` assembly.

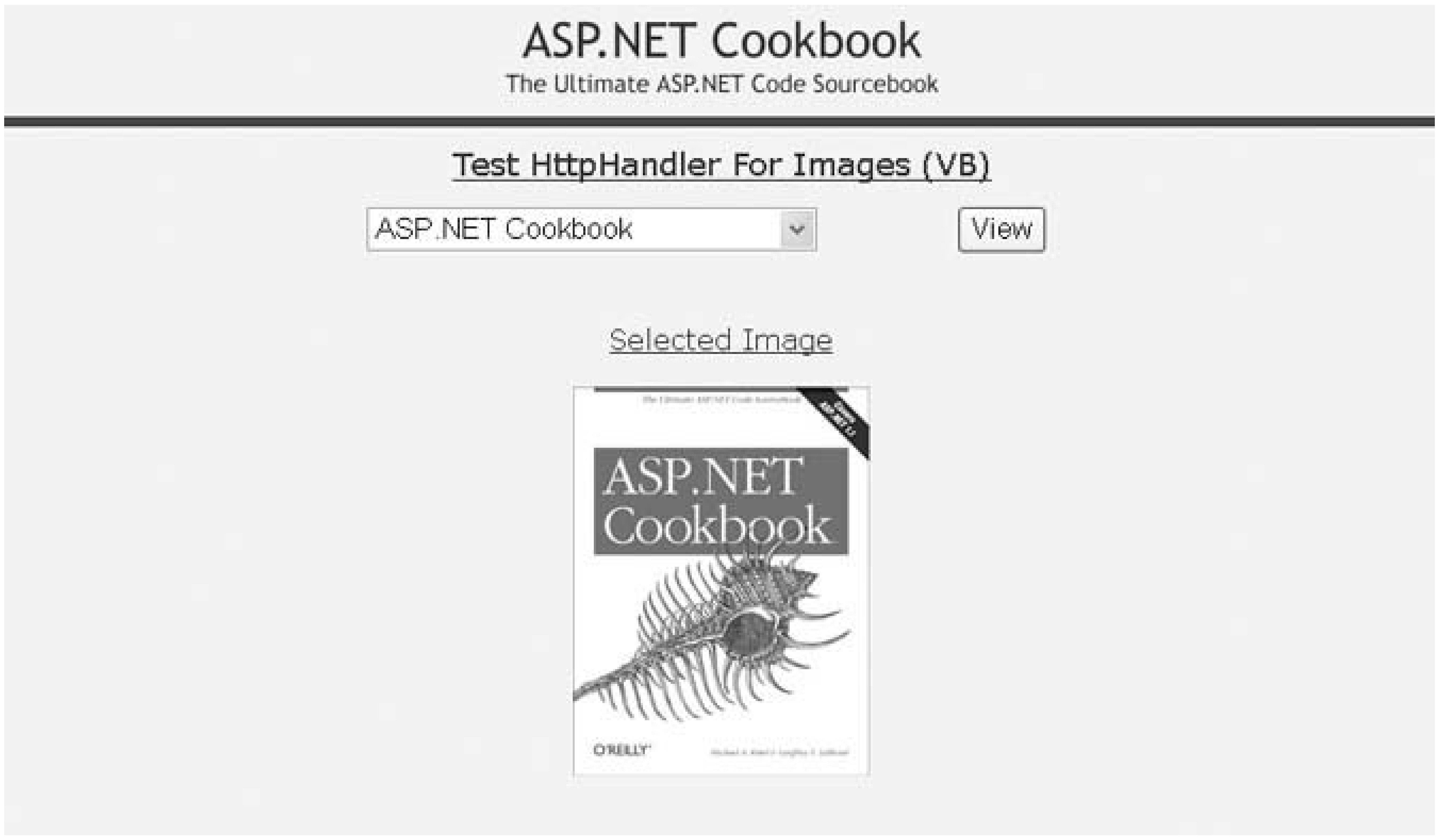
In our example that demonstrates this solution, we have already stored GIF images in a database that can be retrieved and displayed in a browser using our HTTP handler as if they were standard image files. To demonstrate the HTTP handler, we created an ASP.NET page that contains a `DropDownList` , a View button, and an HTML `img` tag. The `DropDownList` displays the descriptions of the images stored in the database. When you make a selection from the list and click the View button, the `src` attribute for the `img` tag is set to the URL of our HTTP handler with the ID of the imag in the URL. When the page is displayed, the browser requests the image from our HTTP handler, which retrieves the ID of the requested image from the URL, reads the data from the database for th image, and streams the image data to the browser. Figure 20-1shows the output of the page used to test our HTTP handler.

The image handler implemented in our example needs several pieces of data to retrieve an image from the database. These include the following:

- The connection string for the database
- The name of the table containing the image data
- The name of the column that uniquely identifies an image (the primary key)
- The name of the column containing the image data
- A unique identifier (ID) for the image that is to be displayed

To be reusable, none of this data can be coded directly into the handler. To get around this problem in our example, we declare four public constants in the image handler class we can use to specify the names of the variables in `Application` scope that contain the database information. In addition, the image ID to be downloaded will be passed in the URL used to access the handler (described later).

Figure 20-1. Output from HTTPHandler test page


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook
Test HttpHandler For Images (VB)
ASP.NET Cookbook
View
Selected Image

The application variables defined by the constants are initialized in the`Application_Start` method of the *global.asax* class, which is executed when an application is started. If you initialize your application variables in the `Application_Start` method, they will always be available when HTTP requests are processed. The code to implement this approach is in Examples 20-1and 20-2 .

To create the image handler, you need to create a class that implements`IHttpHandler` and its two methods, `IsReusable` and `ProcessRequest` . Add the code to process requests made to the handler to the `ProcessRequest` method. As mentioned, `IsReusable` is a property that returns a Boolean value indicating if the HTTP handler can be reused by other HTTP requests. Because our example is a synchronous handler, the property returns `false` so the handler is not pooled (kept in memory).

In our example, the first step in processing a request for an image is to get the ID of the requested image from the URL that is being processed by the handler.

Next, a connection to the database needs to be opened. The connection string is obtained from an `Application` scope variable defined by the `APP_CONNECTION_STR` constant shown in Examples 20-1 (VB) and 20-2 (C#). The name of the database table, along with the columns containing the unique identifier and the image data, are obtained from the `Application` scope variables described earlier. These are used to create the SQL statement required to read the image data from the database.

The next step in our example is to read the image data from the database using the`ExecuteScalar` method of the command object. The `ExecuteScalar` method returns a generic `Object` , so the return value must be cast to the type of data stored in the database. In this case, it must be cast to a byte array.

The image data stored in the database for our example is in GIF format, so the content type is set to "`image/GIF` " to inform the browser of the data type being sent. After setting the content type, the image data is written to the `Response` object using the `BinaryWrite` method.

> If your image is of another type, you will need to set the `ContentType` accordingly. Other choices for images include `image/jpeg`, `image/tiff` , and `image/png` .

To use the handler, we have to add information to the `<httpHandlers>` element of the *web.config* file of the application to tell ASP.NET which URL requests it should route to our custom image handler. You insert this information using an `add` element and its attributes. The `verb` attribute defines the types of requests that are routed to the HTTP handler. The allowable values are *,`GET`, `HEAD` , and `POST` . The value * is a wildcard that specifies that all request types are to be routed to the handler.

The `path` attribute defines the URL(s) that are to be processed by the HTTP handler. The path can be set to a single URL, or to a less specific value such as *.`images` to have the HTTP handler process all requests for URLs with an images extension. In our example, we are setting the path to a specific URL (*ImageHandlerVB.aspx* for the VB example or *ImageHandlerCS.aspx* for the C# example).

> IIS routes requests with the extensions *.asax* , *.ascx* , *.ashx* , *.asmx* , *.aspx* , *.axd* , *.config* ,*.cs* , *.csproj* , *.lic* , *.rem* , *.resources* , *.resx* , *.soap* , *.vb* , *.vbproj* , *.vsdisco* , and *.webinfo* to ASP.NET for processing.
>
> To use an HTTP handler for requests with other extensions, IIS must be configured to send the requests with the desired extensions to the *aspnet_isapi.dll* .

The `type` attribute defines the name of the assembly and class within the assembly that will process the request in the format `type ="`*`class name, assembly`* ". The class name must be identified by its full namespace. Here is the code necessary to add a reference to the image handler to an application *web.config* file:

```
<configuration>
 <system.web>

  …

  <httpHandlers>
   <add verb="*" path="ImageHandlerVB.aspx"
    type="ASPNetCookbook.VBExamples.HttpHandlers.ImageHandlerVB,
    CH20ImageHandlerVB" />
  </httpHandlers>


  …

 </system.web>
</configuration>
```

```
<configuration>
 <system.web>

  …

  <httpHandlers>
   <add verb="*" path="ImageHandlerCS.aspx"
    type="ASPNetCookbook.CSExamples.HttpHandlers.ImageHandlerCS,
    CH20ImageHandlerCS" />
   <add verb="*" path="FileDownloadHandlerCS.aspx"
    type="ASPNetCookbook.CSExamples.HttpHandlers.FileDownloadHandlerCS,
    CH20FileDownloadHandlerCS"/>
  </httpHandlers>

  …

 </system.web>
</configuration>
```

To use the HTTP handler to retrieve images from the database, we need to set the`src` attribute of image tags that will use the HTTP handler to the name of the HTTP handler defined in the`path` attribute of the entry added to *web.config* , passing the ID of the desired image in the URL. In our example, the `src` attribute of an `img` tag is set in the view image button click event of the test page code-behind. Here is a sample URL:

```
 src="ImageHandlerVB.aspx?ImageID=13"
```

The HTTP handler does not have to be implemented in the same language as the application. The C# image handler can be used in VB projects or vice versa.

See Also

Recipe 15.3; search "Introduction to HTTP Handlers" in the MSDN library

Example 20-1. Application variable initialization for image handler (.vb)

```
<%@ Application Language="VB" %>
<%@ Import namespace="ASPNetCookbook.VBExamples.HttpHandlers" %>

<script RunAt="server">

 '''***********************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the application start
 ''' event. It is responsible for initializing application variables.

 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
  Dim strConnection As String

  'get the connection string from web.config
  strConnection = ConfigurationManager. _
   ConnectionStrings("dbConnectionString").ConnectionString

  'Set application variables used in image HTTP handler example
  Application.Add(ImageHandlerVB.APP_CONNECTION_STR, strConnection)
  Application.Add(ImageHandlerVB.APP_IMAGE_TABLE, "BookImage")
  Application.Add(ImageHandlerVB.APP_IMAGE_ID_COLUMN, "BookImageID")
  Application.Add(ImageHandlerVB.APP_IMAGE_DATA_COLUMN, "ImageData")
 End Sub

</script>
```

Example 20-2. Application variable initialization for image handler (.cs)

```
<%@ Application Language="C#" %>
<%@ Import namespace="ASPNetCookbook.CSExamples.HttpHandlers" %>

<script RunAt="server">

 ///***********************************************************************
 /// <summary>
 /// This routine provides the event handler for the application start
 /// event. It is responsible for initializing application variables.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 void Application_Start(Object sender, EventArgs e)
 {
  String strConnection = null;

  // get the connection string from web.config
  strConnection = ConfigurationManager.
   ConnectionStrings["dbConnectionString"].ConnectionString;

  // Set application variables used in image HTTP handler example
  Application.Add(ImageHandlerCS.APP_CONNECTION_STR,
    strConnection);
  Application.Add(ImageHandlerCS.APP_IMAGE_TABLE,
    "BookImage");
  Application.Add(ImageHandlerCS.APP_IMAGE_ID_COLUMN,
    "BookImageID");

  Application.Add(ImageHandlerCS.APP_IMAGE_DATA_COLUMN,
    "ImageData");
 } // Application_Start

</script>
```

Example 20-3. Image HTTP handler (.vb)

```
Option Explicit On
Option Strict On

Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb
Imports System.Web

Namespace ASPNetCookbook.VBExamples.HttpHandlers
```

```
''' <summary>
''' This class provides an image handler as an HTTP handler.
''' </summary>
Public Class ImageHandlerVB
 Implements IHttpHandler

 'The following constant is used in the URL used to access this handler to
 'define the image required
 Public Const QS_IMAGE_ID As String = "ImageID"

 'The following constants define the name of the application variables
 'used to define the database connection string and database table
 'information required to retrieve the required image
 Public Const APP_CONNECTION_STR As String = "DBConnectionStr"
 Public Const APP_IMAGE_TABLE As String = "DBImageTable"
 Public Const APP_IMAGE_ID_COLUMN As String = "DBImageIDColumn"
 Public Const APP_IMAGE_DATA_COLUMN As String = "DBImageDataColumn"

 '''**************************************************************************
 ''' <summary>
 ''' This property defines whether another HTTP handler can reuse this
 ''' instance of the handler.
 ''' </summary>
 '''
 ''' <returns>False</returns>
 ''' <remarks>
 ''' False is always returned since this handler is synchronous and is
 ''' not pooled.
 ''' </remarks>
 Public ReadOnly Property IsReusable() As Boolean _
  Implements IHttpHandler.IsReusable
  Get
   Return (False)
  End Get
 End Property 'IsReusable

 '''**************************************************************************
 ''' <summary>
 ''' This routine provides the processing for the http request. It is
 ''' responsible for reading image data from the database and writing it
 ''' to the response object.
 ''' </summary>
 '''
 ''' <param name="context">Set to the current HttpContext</param>
 Public Sub ProcessRequest(ByVal context As HttpContext) _
  Implements IHttpHandler.ProcessRequest

  Dim dbConn As OleDbConnection = Nothing
  Dim dCmd As OleDbCommand = Nothing
  Dim strConnection As String
  Dim imageTable As String
  Dim imageIDColumn As String
```

```
    Dim imageDataColumn As String
    Dim cmdText As String
    Dim imageID As String
    Dim imageData() As Byte

    Try
     'get the ID of the required image from the querystring
     imageID = context.Request.QueryString(QS_IMAGE_ID)

     'get connection string from application scope and open connection
     'to the database
     strConnection = CStr(context.Application(APP_CONNECTION_STR))
     dbConn = New OleDbConnection(strConnection)
     dbConn.Open()

     'get the name of the database table and columns where the image
     'data is stored then create the SQL to read the data from
     'the database
     imageTable = CStr(context.Application(APP_IMAGE_TABLE))
     imageIDColumn = CStr(context.Application(APP_IMAGE_ID_COLUMN))
     imageDataColumn = CStr(context.Application(APP_IMAGE_DATA_COLUMN))

     cmdText = "SELECT " & imageDataColumn & _
       " FROM " & imageTable & _
       " WHERE " & imageIDColumn & "=?"

     dCmd = New OleDbCommand(cmdText, dbConn)
     dCmd.Parameters.Add(New OleDbParameter("ImageID", _
              imageID))

     'get the image data
     imageData = CType(dCmd.ExecuteScalar(), Byte())

     'write the image data to the reponse object
     context.Response.ContentType = "image/gif"
     context.Response.BinaryWrite(imageData)

    Finally
     'clean up
     If (Not IsNothing(dbConn)) Then
     dbConn.Close()
     End If
    End Try
   End Sub 'ProcessRequest
  End Class 'ImageHandlerVB
End Namespace
```

Example 20-4. Image HTTP handler (.cs)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;
using System.Web;

namespace ASPNetCookbook.CSExamples.HttpHandlers
{
 /// <summary>
 /// This class provides an image handler as an HTTP handler.
 /// </summary>
 public class ImageHandlerCS : IHttpHandler
 {
  // The following constant is used in the URL used to access this handler
  // to define the image required
  public const string QS_IMAGE_ID = "ImageID";

  // The following constants defines the name of the application variables
  // used to define the database connection string and database table
  // information required to retrieve the required image
  public const string APP_CONNECTION_STR = "DBConnectionStr";
  public const string APP_IMAGE_TABLE = "DBImageTable";
  public const string APP_IMAGE_ID_COLUMN = "DBImageIDColumn";
  public const string APP_IMAGE_DATA_COLUMN = "DBImageDataColumn";

  ///***********************************************************************
  /// <summary>
  /// This property defines whether another HTTP handler can reuse this
  /// instance of the handler.
  /// </summary>
  ///
  /// <returns>false</returns>
  /// <remarks>
  /// false is always returned since this handler is synchronous and is
  /// not pooled.
  /// </remarks>
  public bool IsReusable
  {
   get
   {

    return (false);
   }
  } // IsReusable

  ///***********************************************************************
  /// <summary>
  /// This routine provides the processing for the http request. It is
  /// responsible for reading the file from the local filesystem and
  /// writing it to the response object.
  /// </summary>
```

```
///
/// <param name="context">Set to the current HttpContext</param>
public void ProcessRequest(HttpContext context)
{
 OleDbConnection dbConn = null;
 OleDbCommand dCmd = null;
 String strConnection = null;
 String imageTable = null;
 String imageIDColumn = null;
 String imageDataColumn = null;
 String strSQL = null;
 String imageID = null;
 byte[] imageData = null;

 try
 {
  // get the ID of the required image from the querystring
  imageID = context.Request.QueryString[QS_IMAGE_ID];

  // get connection string from application scope and open connection
  // to the database
  strConnection = (String)(context.Application[APP_CONNECTION_STR]);
  dbConn = new OleDbConnection(strConnection);
  dbConn.Open();

  // get the name of the database table and columns where the image
  // data is stored then create the SQL to read the data from
  // the database
  imageTable = (String)(context.Application[APP_IMAGE_TABLE]);
  imageIDColumn = (String)(context.Application[APP_IMAGE_ID_COLUMN]);
  imageDataColumn = (String)(context.Application[APP_IMAGE_DATA_COLUMN]);

  strSQL = "SELECT " + imageDataColumn +
   " FROM " + imageTable +
   " WHERE " + imageIDColumn + "=?";

  dCmd = new OleDbCommand(strSQL, dbConn);
  dCmd.Parameters.Add(new OleDbParameter("ImageID",
         imageID));
  imageData = (byte[])(dCmd.ExecuteScalar());

  // write the image data to the reponse object
  context.Response.ContentType = "image/gif";
  context.Response.BinaryWrite(imageData);
 } // try

 finally
 {
  // clean up
  if (dbConn != null)
  {
  dbConn.Close();
```

```
    }
   } // finally
  } // ProcessRequest
 } // ImageHandlerCS
}
```

Example 20-5. Using the image HTTP handler (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH20TestHTTPImageHandlerVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH20TestHTTPImageHandlerVB"
 Title="Test HTTP Image Handler" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Test HttpHandler For Images (VB)
 </div>
 <table width="50%" align="center" border="0">
  <tr>
   <td>
    <asp:DropDownList ID="ddImages" Runat="server" />
   </td>
   <td>
    <input id="btnViewImage" runat="server"
    type="button"
    value="View"
    onserverclick="btnViewImage_ServerClick" />
   </td>
  </tr>
  <tr>
   <td id="tdSelectedImage" runat="server"
     colspan="2"
    align="center"
    class="subHeading">
    <br /><br />
    Selected Image<br /><br />
    <img id="imgBook" runat="server" border="0" alt="book" src=""/>
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 20-6. Using the image HTTP handler code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports ASPNetCookbook.VBExamples.HttpHandlers
Imports System.Configuration
Imports System.Data
Imports System.Data.Common
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH20TestHTTPImageHandlerVB.aspx
 ''' </summary>
 Partial  Class  CH20TestHTTPImageHandlerVB
  Inherits System.Web.UI.Page
 '''*************************************************************************
 ''' <summary>
 ''' This routine provides the event handler for the page load event. It
 ''' is responsible for initializing the controls on the page.
 ''' </summary>
 '''
 ''' <param name="sender">Set to the sender of the event</param>
 ''' <param name="e">Set to the event arguments</param>
 Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
  Dim dbConn As OleDbConnection = Nothing
  Dim dc As OleDbCommand
  Dim dr As OleDbDataReader
  Dim strConnection As String
  Dim strSQL As String

  If (Not Page.IsPostBack) Then
   Try
    'initially hide the selected image since one is not selected
    tdSelectedImage.Visible = False

    'get the connection string from web.config and open a connection
    'to the database
    strConnection = ConfigurationManager. _
    ConnectionStrings("dbConnectionString").ConnectionString
    dbConn = New OleDbConnection(strConnection)
    dbConn.Open()

    'build the query string and get the data from the database
    strSQL = "SELECT BookImageID, Title " & _
    "FROM BookImage " & _
    "ORDER BY Title"
    dc = New OleDbCommand(strSQL, dbConn)
    dr = dc.ExecuteReader()
    'set the source of the data for the repeater control and bind it
    ddImages.DataSource = dr
    ddImages.DataTextField = "Title"
    ddImages.DataValueField = "BookImageID"
    ddImages.DataBind()

    Finally
    'clean up
    If (Not IsNothing(dbConn)) Then
    dbConn.Close()
    End If
    End Try
   End If
  End Sub 'Page_Load

  '''**************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the view image click
  ''' event. It is responsible for setting the src attibute of the
  ''' imgBook tag to the URL of the HTTP handler that will deliver the
  ''' image content.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnViewImage_ServerClick(ByVal sender As Object, _
        ByVal e As System.EventArgs)
   'set the source for the selected image tag
   imgBook.Src = "ImageHandlerVB.aspx?" & _
      ImageHandlerVB.QS_IMAGE_ID & "=" & _
      ddImages.SelectedItem.Value.ToString()

   'make the selected image visible
   tdSelectedImage.Visible = True
  End Sub 'btnViewImage_ServerClick
 End Class 'CH20TestHTTPImageHandlerVB
End Namespace
```

Example 20-7. Using the image HTTP handler code-behind (.cs)

```
using ASPNetCookbook.CSExamples.HttpHandlers;
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;

namespace ASPNetCookbook.CSExamples
```

```
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH20TestHTTPImageHandlerCS.aspx
 /// </summary>

 public partial class CH20TestHTTPImageHandlerCS  :  System.Web.UI.Page
 {
  ///*************************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   OleDbConnection dbConn = null;
   OleDbCommand dc;
   OleDbDataReader dr;
   String strConnection;
   String strSQL;

   if (!Page.IsPostBack)
   {
    try
    {
    // initially hide the selected image since one is not selected
    tdSelectedImage.Visible = false;

    // get the connection string from web.config and open a connection
    // to the database
    strConnection = ConfigurationManager.
    ConnectionStrings["dbConnectionString"].ConnectionString;
    dbConn = new OleDbConnection(strConnection);
    dbConn.Open();

    // build the query string and get the data from the database
    strSQL = "SELECT BookImageID, Title " +
     "FROM BookImage " +
     "ORDER BY Title";
    dc = new OleDbCommand(strSQL, dbConn);
    dr = dc.ExecuteReader();

    // set the source of the data for the repeater control and bind it
    ddImages.DataSource = dr;
    ddImages.DataTextField = "Title";
    ddImages.DataValueField = "BookImageID";
    ddImages.DataBind();

    }
    finally
    {
    // clean up
    if (dbConn != null)
    {
    dbConn.Close();

    }
    }
   }
  } // Page_Load

  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the view image click
  /// event. It is responsible for setting the src attibute of the
  /// imgBook tag to the URL of the HTTP handler that will deliver the
  /// image content.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void btnViewImage_ServerClick(Object sender,
         System.EventArgs e)
  {
   // set the source for the selected image tag
   imgBook.Src = "ImageHandlerCS.aspx?" +
      ImageHandlerCS.QS_IMAGE_ID + "=" +
      ddImages.SelectedItem.Value.ToString();

   // make the selected image visible
   tdSelectedImage.Visible = true;
  } // btnViewImage_ServerClick
 } // CH20TestHTTPImageHandlerCS
}
```

Recipe 20.3. Creating a File Download Handler

Problem

You want to create a general-purpose file handler to downloadfiles of any type where the name of the file being downloaded is provided at runtime.

Solution

Create an HTTP handler to read the required file from the filesystem and send it to the browser. The steps for creating an HTTP handler are defined in Recipe 20.1.

Examples 20-8 and 20-9 show the VB and C# class files we've written to implement a file download HTTP handler. Examples 20-10, 20-11 through 20-12 show the *.aspx* file and the VB and C# code-behind files for our application that demonstrates the use of the file download HTTP handler.

Discussion

Of the many ways you might implement a reusable file download routine that can handle almost any file type, creating an HTTP handler makes the most sense. HTTP handlers are designed to process requests for resources, and a file download request is a special instance of such a request.

The file download HTTP handler described in our example downloads a file from the localfilesystem to the user's system. The name of the file to download is passed in the URL used to access the HTTP handler.

The first step in implementing the file download HTTP handler is to create a class that implements `IHttpHandler`. The class can be part of your web project, or if you want it to be reusable across applications, you can place it in a project by itself so a separate assembly can be created, as described in Recipe 20.1.

As discussed in the previous recipe, implementing the`IHttpHandler` interface requires the implementation of two methods: `IsReusable` and `ProcessRequest`. `IsReusable` is a property that explicitly returns a Boolean value that indicates if the HTTP handler can be reused by other HTTP requests. For synchronous handlers like our example, the property should always return`false` so the handler is not pooled (kept in memory).

The `ProcessRequest` method provides all of the functionality required to download the file. The first step is to retrieve the name of the file that is to be downloaded from the URL that is being processed by the handler.

In our example, we provide only the filename in the URL that calls the handler because all files to be downloaded are intended to be located in a single *Downloads* directory associated with the application. However, because a fully qualified filename is required for the download along with the size of the file, a `FileInfo` object is created passing the fully qualified name of the file to the constructor. The path to the *Downloads* directory is obtained using `Server.MapPath`, which translates a relative path within the web application to a fully qualified path on the web server's filesystem.

> Review how you've implemented your file download handler to ensure a hacker cannot download files you do not intend to have downloaded. For example, if you do not restrict downloadable files to a single area, as we have done in our example, a hacker could possibly enter the following to download your *web.config* file:
>
> http://aspnetcookbook/FileDownloadHandlerVB.aspx?*filename=web.config*
>
> By restricting downloadable files to a single area and providing the path information in your code instead of in the URL, you can block a hacker from accessing restricted folders. In addition, you can use filesystem security to improve the security of your application.

To send a file to the browser, you must write it to the `Response` object. Passing data in the response object is the only way to return data to the browser. Based on the content type (described later), the browser processes the data returned using the `Response` object.

The first step in writing the data to the `Response` object is to clear any data in the object because no other data can be included with the file or a corrupted file error will occur.

The `AddHeader` method of the `Response` object is then used to add the name of the file being downloaded and its length.

The content type then needs to be set. In our example, it is set to `application/octet-stream` so it will be treated by the browser as a binary stream and prompt the user to select the location to save the file. For your application, you may want to set the content type to the explicit file type, such as `application/PDF` or `application/msword`. Setting the content type to the explicit file type allows the browser to open it with the application defined to handle the specified file type on the client machine. For more information on content types, consult http://www.w3c.org.

The file is written to the `Response` object and the response is ended to send the file to the browser.

The *web.config* file for our application must have an entry in the `<httpHandlers>` element to tell ASP.NET which URL requests need to be routed to the file downloadHTTP handler. You use the `add` element and its attributes to specify each custom handler. The `verb` attribute of the `add` element defines the types of requests that are routed to the HTTP handler. The allowable values are `*`, `GET`, `HEAD`, and `POST`. The value `*` is a wildcard for all request types.

The `path` attribute defines the URL(s) that are to be processed by the HTTP handler. The path can be a single URL, as shown later, or it can be set to something like `*download` to have the HTTP handler process all requests for URLs with a `download` extension. (See the note in Recipe 20.1 regarding request extensions.)

The `type` attribute defines the name of the assembly and class within the assembly that will process the request in the format `type="class name, assembly"`. The class must be identified by its full namespace. The following code shows how to add a reference to thefile download handler to the *web.config* file of our sample application:

```
<configuration>
 <system.web>

   …

   <httpHandlers>
    <add verb="*" path="FileDownloadHandlerVB.aspx"
    type="ASPNetCookbook.VBExamples.HttpHandlers.FileDownloadHandlerVB,
    CH20FileDownloadHandlerVB"/>
   </httpHandlers>
```

```
   …

 </system.web>
</configuration>
```

```
<configuration>
 <system.web>

   …

   <httpHandlers>
    <add verb="*" path="FileDownloadHandlerCS.aspx"
    type="ASPNetCookbook.CSExamples.HttpHandlers.FileDownloadHandlerCS,
    CH20FileDownloadHandlerCS"/>
   </httpHandlers>

   …

 </system.web>
</configuration>
```

To use the HTTP handler (described earlier) to download files, we need to set the `href` attribute of an HTML anchor tag to the name of the file download HTTP handler defined in the path attribute of the entry added to *web.config*, passing the name of the file to download in the URL. In our example, the `HRef` attribute of the anchor tag is set in the `Page_Load` method of the test page. Here is a sample of the URL:

```
href="FileDownloadHandlerVB.aspx?Filename=SampleDownload.txt"
```

This example demonstrates downloading a file that exists on the web server. The code can be alterec to pass additional data in the URL, programmatically generate the file, and send it to the browser without saving it to the filesystem. This could be useful if you are dynamically creating a PDF or a CSV file requested by the user.

See Also

Recipe 20.1 for techniques on how to create a generic, reusable HTTP handler;http://www.w3c.org for information on content types

Example 20-8. File download HTTP handler (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.IO
Imports System.Web

Namespace ASPNetCookbook.VBExamples.HttpHandlers
 ''' <summary>
 ''' This class provides a file download handler as an HTTP handler.
 ''' </summary>
 Public Class FileDownloadHandlerVB
  Implements IHttpHandler

  'The following constant is used in the URL used to access this handler to
  'define the file to download
  Public Const QS_FILENAME As String = "Filename"

  'the following constant defines the folder containing downloadable files
  Private Const DOWNLOAD_FOLDER As String = "Downloads"
  '''*****************************************************************************
  ''' <summary>
  ''' This property defines whether another HTTP handler can reuse this
  ''' instance of the handler.
  ''' </summary>
  '''
  ''' <returns>False</returns>
  ''' <remarks>
  ''' False is always returned since this handler is synchronous and is
  ''' not pooled.
  ''' </remarks>
  Public ReadOnly Property IsReusable() As Boolean _
   Implements IHttpHandler.IsReusable
```

```
    Get
     Return (False)
    End Get
   End Property 'IsReusable

   '''*********************************************************************
   ''' <summary>
   ''' This routine provides the processing for the http request. It is
   ''' responsible for reading the file from the local filesystem and
   ''' writing it to the response object.
   ''' </summary>
   '''
   ''' <param name="context">Set to the current HttpContext</param>
   Public Sub ProcessRequest(ByVal context As HttpContext) _
    Implements IHttpHandler.ProcessRequest

    Dim file As FileInfo
    Dim filename As String

    'get the filename from the querystring
    filename = context.Request.QueryString(QS_FILENAME)

    'get the file data since the length is required for the download
    file = New FileInfo(context.Server.MapPath(DOWNLOAD_FOLDER) & "\" & _
     filename)

    'write it to the browser
    context.Response.Clear()
    context.Response.AddHeader("Content-Disposition", _
      "attachment; filename=" & filename)
    context.Response.AddHeader("Content-Length", _
       file.Length.ToString())

    context.Response.ContentType = "application/octet-stream"
    context.Response.WriteFile(file.FullName)
    context.Response.End()
   End Sub 'ProcessRequest
  End Class 'FileDownloadHandlerVB
End Namespace
```

Example 20-9. File download HTTP handler (.cs)

```
using System;
using System.IO;
using System.Web;

namespace ASPNetCookbook.CSExamples.HttpHandlers
```

```
{
 /// <summary>
 /// This class provides a file download handler as an HTTP handler.
 /// </summary>
 public class FileDownloadHandlerCS : IHttpHandler
{
  // The following constant is used in the URL used to access this handler
  // to define the file to download
  public const string QS_FILENAME = "Filename";

  //the following constant defines the folder containing downloadable files
  private const string DOWNLOAD_FOLDER = "Downloads";

  ///*************************************************************************
  /// <summary>
  /// This property defines whether another HTTP handler can reuse this
  /// instance of the handler.
  /// </summary>
  ///
  /// <returns>false</returns>
  /// <remarks>
  /// false is always returned since this handler is synchronous and is
  /// not pooled.
  /// </remarks>
  public bool IsReusable
  {
  get
  {
   return (false);
  }
  } // IsReusable

  ///*************************************************************************
  /// <summary>
  /// This routine provides the processing for the http request. It is
  /// responsible for reading the file from the local filesystem and
  /// writing it to the response object.


  /// </summary>
  ///
  /// <param name="context">Set to the current HttpContext</param>
  public void ProcessRequest(HttpContext context)
  {
   FileInfo file = null;
   string filename = null;

   // get the filename from the querystring
   filename = context.Request.QueryString[QS_FILENAME];

   // get the file data since the length is required for the download
   file = new FileInfo(context.Server.MapPath(DOWNLOAD_FOLDER) + "\\" +
       filename);

   // write it to the browser
   context.Response.Clear();
   context.Response.AddHeader("Content-Disposition",
     "attachment; filename=" + filename);
   context.Response.AddHeader("Content-Length",
      file.Length.ToString());
   context.Response.ContentType = "application/octet-stream";
   context.Response.WriteFile(file.FullName);
   context.Response.End();
  } // ProcessRequest
 } // FileDownloadHandlerCS
}
```

Example 20-10. Using the file download HTTP handler (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH20TestHTTPFileDownloadHandlerVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH20TestHTTPFileDownloadHandlerVB"
 Title="Test HTTP File Download Handler" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  HTTP Handler For File Downloads (VB)
 </div>
 <div align="center" class="subHeading">
  <br />
  Click the link below to download the sample file using an HTTP Handler
  <br />
  <br />
  <a id="anDownload" runat="server" ></a>
 </div>
</asp:Content>
```

Example 20-11. Using the file download HTTP handler code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports ASPNetCookbook.VBExamples.HttpHandlers

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH20TestHTTPFileDownloadHandlerVB.aspx
 ''' </summary>
 Partial Class CH20TestHTTPFileDownloadHandlerVB
  Inherits System.Web.UI.Page
  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
   Const FILE_TO_DOWNLOAD As String = "SampleDownload.txt"

   'set the text and href of the download anchor
   anDownload.InnerText = FILE_TO_DOWNLOAD
   anDownload.HRef = "FileDownloadHandlerVB.aspx?" & _
    FileDownloadHandlerVB.QS_FILENAME & "=" & _
    FILE_TO_DOWNLOAD
  End Sub 'Page_Load
 End Class 'CH20TestHTTPFileDownloadHandlerVB
End Namespace
```

Example 20-12. Using the file download HTTP handler code-behind (.cs)

```
using ASPNetCookbook.CSExamples.HttpHandlers;
using System;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH20TestHTTPFileDownloadHandlerCS.aspx
 /// </summary>
 public partial class CH20TestHTTPFileDownloadHandlerCS : System.Web.UI.Page
 {
  ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)

  {
   const string FILE_TO_DOWNLOAD = "SampleDownload.txt";
   // set the text and href of the download anchor
   anDownload.InnerText = FILE_TO_DOWNLOAD;
   anDownload.HRef = "FileDownloadHandlerCS.aspx?" +
    FileDownloadHandlerCS.QS_FILENAME + "=" +
    FILE_TO_DOWNLOAD;
  } // Page_Load
 } //  CH20TestHTTPFileDownloadHandlerCS
}
```

PREV

Chapter 21. Assorted Tips

21.0 Introduction

This chapter contains an assortment of recipes that don't fit conveniently into the other chapters of the book. If there is a common theme among them, it is that they illustrate that the .NET Framework's class libraryspecifically the classes that support ASP.NETis more expansive than you might expect. Indeed, ASP.NET provides the functionality to support almost everything an applicatior needs. You just have to be willing to dig deep enough to find it.

PREV

Recipe 21.2. Accessing HTTP-Specific Information from Within a Class

Problem

You want to create a business service class that can be used by any page in your site, and you want it to have access to the HTTP-specific information available in web pagesthat is, all the server objects used by the application.

Solution

Add a reference to the `System.Web` assembly in your business service project and a companion `Imports` statement (or `using` statement in C#) to your class, and use the `Current` property of the `HttpContext` object to access the desired server objects.

In the business service class, use the .NET language of your choice to:

1. Add a reference to `System.Web`.
2. Import the `System.Web` namespace.
3. Reference the current HTTP context when accessing server objects, as in `HTTPContext.Current.Session`.

Examples 21-1 and 21-2 show the VB and C# class files for an example business service that implements this solution.

Discussion

By referencing the `Current` property of the `HttpContext` object in the business class, your code has full access to all the server objects used in web applications. This includes the ability to access all information about the request being made, the response being returned, session data, and application data. For more information, refer to the `HttpContext` class in the MSDN documentation.

When you create an ASP.NET application with Visual Studio .NET, all the pages and classes of the web project have access to the HTTP-specific information. This is because Visual Studio automatically adds a reference to `System.Web` when you create the project. When you create a Class Library project, the technique commonly used to create reusable assemblies, Visual Studio does not automatically add the reference to `System.Web`. You can do so manually, however, by right-clicking the project in the Solution Explorer and selecting Add Reference from the context menu. When the

Add Reference dialog box appears, choose the *System.Web.dll* component from the .NET tab.

If you are not using Visual Studio for your project, you can add the reference as part of the compile command. The following fragments show how we do this for our example:

```
vbc /target:library /reference:System.Web.dll /out:BusinessService.dll
  BusinessService.vb
```

```
csc /target:library /reference:System.Web.dll /out:BusinessService.dll
  BusinessService.cs
```

In the class requiring access to the HTTP-specific information, add an `Imports` statement (or `using` statement in C#) for the `System.Web` assembly. Adding this statement imports the namespace and provides access to the HTTP objects without having to fully qualify each reference to the object, making the code more readable and easier to maintain. The difference in access methods is shown here:

```
'access to the Session object when Imports statement is included
value = CStr(HttpContext.Current.Session("someData"))

'access to the Session object when the Imports statement is NOT included
value = CStr(System.Web.HttpContext.Current.Session("someData"))
```

```
// access to the Session object when Imports statement is included
value = (string)(HttpContext.Current.Session["someData"]);

// access to the Session object when the Imports statement is NOT included
value = (string)(System.Web.HttpContext.Current.Session["someData"]);
```

By adding the reference and `Imports` (or `using`) statement to your business service projects, your code has full access to all the server objects used in web applications.

Providing access to the server objects in your business classes is useful and necessary for some services, but it should not be universally applied. When your business service requires access to the HTTP-specific information, it can no longer be used in non-web applications, reducing its reusability.

See Also

HttpContext class documentation in the MSDN Library

Example 21-1. Accessing HTTP-specific information (.vb)

```
Imports System.Web

Public Class BusinessService

  Public Sub doSomething()
 Dim value As String

 value = CStr(HttpContext.Current.Session("someData"))

 'use the data from session as required
  End Sub
End Class 'BusinessService
```

Example 21-2. Accessing HTTP-specific information (.cs)

```
using System.Web;

namespace CSMisc
{
  public class BusinessService
  {
 public BusinessService()
 {
   string value;

   value = (string)(HttpContext.Current.Session["someData"]);

   // use the data from session as required
 }
  }
}
```

PREV

Recipe 21.3. Executing External Applications

Problem

You need to run an external application from your web application to perform a required operation.

Solution

Use the `System.Diagnostics.Process.Start` method to call your external application.

In the code-behind class for the page, use the .NET language of your choice to:

1. Import the `System.Diagnostics` namespace.
2. Create a `ProcessStartInfo` object, passing the name of the external application to run along with any required command-line parameters.
3. Set the working directory to the location of the external application.
4. Start the external application process by calling the `Start` method of the `Process` class, passing the `ProcessStartInfo` object.

Examples 21-3 and 21-4 show the relevant portion of the sample VB and C# codebehind files that illustrate this solution.

Discussion

Applications frequently must interface with other applications or systems that use different technologies. At the same time, it may be impractical to migrate these applications to the new platforms or to provide web service wrappers to gain access to the applications. Sometimes, the only practical solution is to execute another program to perform the required operation. For example, you may have an existing application that exports data from your COBOL accounting program to a forma usable by other systems. The common language runtime (CLR) provides a set of classes to support running other applications from within the .NET environment. These classes are part of the `System.Diagnostics` assembly.

The first step to running an external application from within ASP.NET applications is to create a `ProcessStartInfo` object and to pass it the name of the application to run along with any command-line parameters it might require. In our example, we use the Java runtime to execute a Java progran called `AJavaProgram`. In our case, the name of the application to run is *java* and the name of the Java program to run, *AJavaProgram*, is the only required command-line parameter.

VB

```
 si = New ProcessStartInfo("java", _
      "AJavaProgram")
```

C#

```
 si = new ProcessStartInfo("java",
      "AJavaProgram");
```

Next, the working directory is set to the location of the Java application. For this example, the Java application (`AJavaProgram.class`) is located in the root directory of the ASP.NET application, so `Server.MapPath` is passed . to get the fully qualified path to the root directory:

VB

```
 si.WorkingDirectory = Server.MapPath(".")
```

C#

```
 si.WorkingDirectory = Server.MapPath(".");
```

The `ProcessStartInfo` class has been changed in Version 2.0 of the .NET Framework with the addition of the `UserName` and `Password` properties, along with a modification in the use of the `UseShellExecute` property. In Version 2.0, if the `UserName` property is set to `Nothing (null` in C#), the `UseShellExecute` property must be set to `false`, or an `InvalidOperation-Exception` will be thrown. Refer to the `ProcessStartInfo` class documentation in the MSDN Library for more information.

The application is then started by calling the `Start` method of the `Process` class passing the `ProcessStartInfo` object containing the application information. The `Start` method is shared (or static), which does not require instantiating a `Process` object.

```
 proc = Process.Start(si)
```

```
 proc = Process.Start(si);
```

To wait for the process to complete before continuing execution of the ASP.NET application, the `WaitForExit` method is called, optionally passing a maximum time to wait. Once the `WaitForExit` method is called, page execution is paused until the process completes or the timeout occurs. If you do not need to wait on the process to complete, calling the `WaitForExit` method will be unnecessary.

VB

```
proc.WaitForExit()
```

C#

```
proc.WaitForExit();
```

If the process takes longer to complete than the passed timeout value, the process will not be terminated; it will continue until completion. If you do not want the process to continue executing, your application can terminate it by calling the `kill` method of the process object.

By default, external applications are run using the ASP.NET user account. As a result, you may need to change the permissions for the ASP.NET account depending on the operations theexternal application performs. Take care when giving the ASP.NET user additional permissions to avoid creating security problems on your server.

The user account used by ASP.NET is defined in the `userName` attribute of the `processModel` element in the *machine.config* file. By default, the `userName` attribute is set to `machine`, which is a special username indicating the ASP.NET user. The `userName` can be set to any local or domain username that you want ASP.NET to run under.

See Also

`ProcessStartInfo` class documentation in the MSDN Library

Example 21-3. Running an external application (.vb)

```
Imports System.Diagnostics

  …

Dim proc As Process
Dim si As ProcessStartInfo

'create a new start information object with the program to execute
'and the command line parameters
si = New ProcessStartInfo("java", _
      "AJavaProgram")

'set the working directory where the external program is located
si.WorkingDirectory = Server.MapPath(".")
si.UseShellExecute = False

'start a new process using the start information object
proc = Process.Start(si)

'wait for process to complete before continuing
proc.WaitForExit()
```

Example 21-4. Running an external application (.cs)

```
using System;
using System.Diagnostics;

  …

Process proc = null;
ProcessStartInfo si = null;

// create a new start information object with the program to execute
// and the command line parameters
si = new ProcessStartInfo("java",
      "AJavaProgram");

// set the working directory where the external program is located
si.WorkingDirectory = Server.MapPath(".");
si.UseShellExecute = false;

// start a new process using the start information object
proc = Process.Start(si);

// wait for process to complete before continuing
proc.WaitForExit();
```

PREV

Recipe 21.4. Transforming XML to HTML

Problem

The content for your application is in XML format, and you need to transform it to HTML for display in a browser.

Solution

Use an ASP.NET XML control and set its `DocumentSource` property to the XML document you need to transform and the `transformSource` property to the XSLT document that specifies the transformation to be performed.

In the *.aspx* file, place an `asp:Xml` control where you want the HTML from the transformation to be placed in the page.

In the code-behind class for the page, use the .NET language of your choice to:

1. Set the `DocumentSource` property of the XML control to the relative path to the XML document to convert.
2. Set the `transformSource` property to the relative path to the XSLT document.

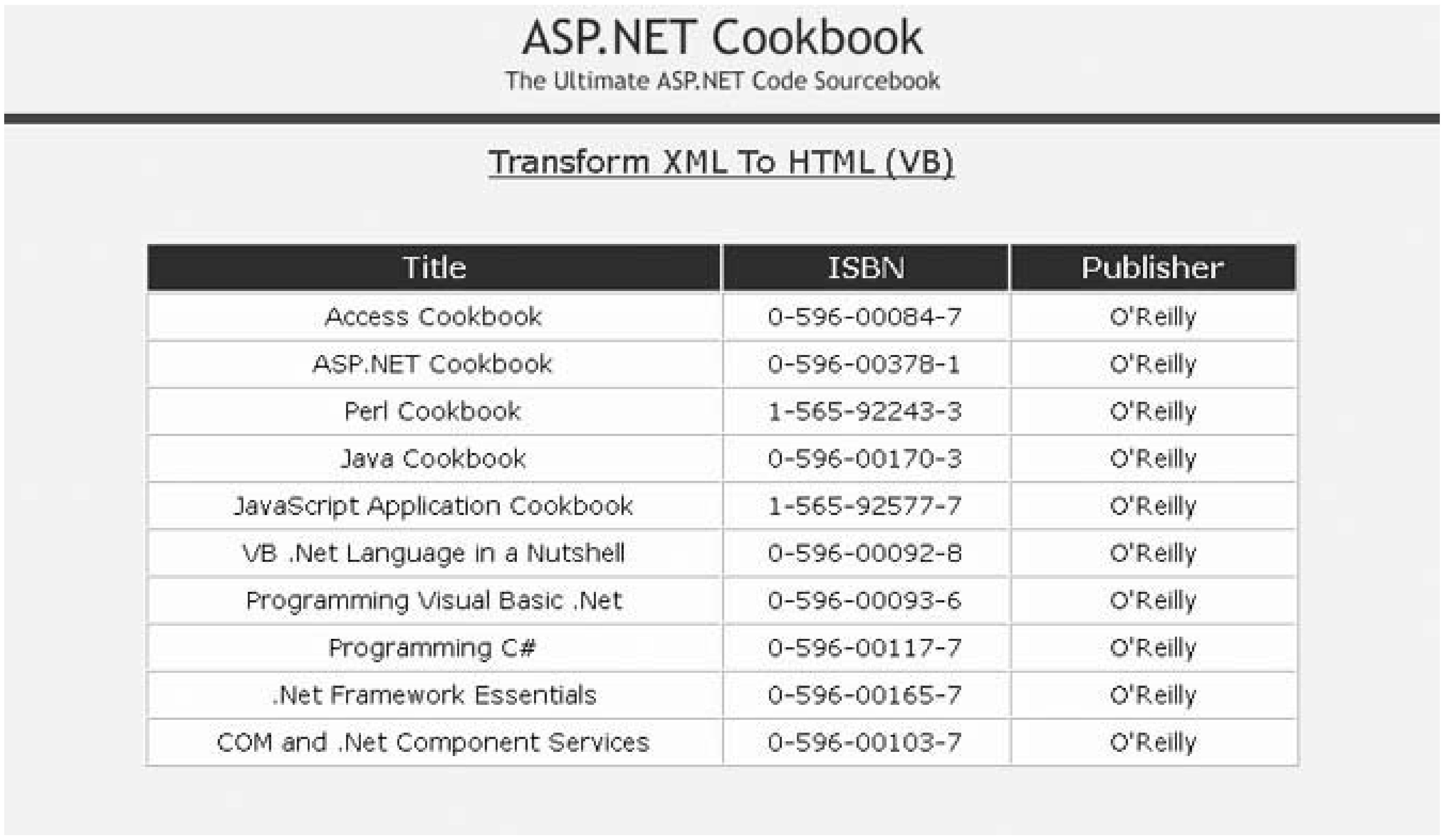
Examples 21-5 , 21-6 through 21-7 show the *.aspx* file and VB and C# code-behind files for an application that demonstrates this solution. The XML used as the source is shown in Example 21-8, and the XSLT used to transform the XML is shown in Example 21-9. The output transformed to HTML is shown in Figure 21-1 .

Discussion

XML is becoming the predominant format for storing content. XML provides a platform-independent format that can be converted to many other formats, including HTML. By storing the content for your web application in XML, the same content can be transformed to the HTML needed for display in a standard browser or the HTML needed for a PDA.

In our example to illustrate this solution, an XML document containing book information (Example 21-8) is transformed into an HTML table using an XSLT document (Example 21-9). The transformation is performed by an `Xml` server control.

Figure 21-1. Transforming XML to HTML output


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Transform XML To HTML (VB)

Title	ISBN	Publisher
Access Cookbook	0-596-00084-7	O'Reilly
ASP.NET Cookbook	0-596-00378-1	O'Reilly
Perl Cookbook	1-565-92243-3	O'Reilly
Java Cookbook	0-596-00170-3	O'Reilly
JavaScript Application Cookbook	1-565-92577-7	O'Reilly
VB .Net Language in a Nutshell	0-596-00092-8	O'Reilly
Programming Visual Basic .Net	0-596-00093-6	O'Reilly
Programming C#	0-596-00117-7	O'Reilly
.Net Framework Essentials	0-596-00165-7	O'Reilly
COM and .Net Component Services	0-596-00103-7	O'Reilly

When you use an `Xml` control to do the work, you need to set only two of its properties to convert XML to HTML. The `DocumentSource` property needs to be set to the relative path to the XML document to convert, and the `TRansformSource` property needs to be set to the relative path to the XSLT document, as we have done in our example:

```
xmlTransform.DocumentSource = "xml/books.xml"
xmlTransform.TransformSource = "xml/books.xslt"
```

```
xmlTransform.DocumentSource = "xml//books.xml";
xmlTransform.TransformSource = "xml//books.xslt";
```

Most controls that use files require a fully qualified name of the file on the web server. However, the `Xml` control requires a relative path from the root folder of the web site to the XML and XSLT files. Setting the properties to fully qualified paths will result in an exception being thrown.

The majority of the work to perform an XSL transformation is in the creation of the XSLT. An XSLT document is a specially formatted XML document and, as such, requires a declaration defining the version of XML and the encoding of the document.

In addition, elements indicating that the document is an XSL`stylesheet` and defining the output method are required:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet  version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  …

</xsl:stylesheet>
```

In XSLT, templates are used to replace specific content in the XML document with the template defined in an `xsl:template` element. In our example, two templates are used. The first template defines the base structure of the HTML table that will be used to display the XML content when it is converted to HTML, as shown here:

```
<!-- output main table with header row -->
<xsl:template match="Root">
 <table width="80%" border="1" cellspacing="0" cellpadding="4" align="center">
  <tr class="tableHeader" bgcolor="#000080">
   <td width="50%" align="center" >Title</td>
   <td width="25%" align="center" >ISBN</td>
   <td width="25%" align="center" >Publisher</td>
  </tr>
  <xsl:apply-templates select="Book" />
 </table>
</xsl:template>
```

This template instructs the conversion to apply additional templates using the`Book` element of the XML document as a source for the data:

```
<xsl:apply-templates select="Book"/>
```

The `xsl:apply-templates` element is roughly equivalent to a `for` loop that would iterate through each `Book` element in the XML document applying our second template:

```
<!-- output a row in the table for each Book node in the XML document -->
<xsl:template match="Book" >
 <tr class="tableCellNormal" bgcolor="#FFFFE0">
   <td width="50%">
    <xsl:value-of select="Title"/>
  </td>
   <td width="25%" align="center">
```

```
    <xsl:value-of select="ISBN" />
   </td>
   <td width="25%" align="center">
    <xsl:value-of select="Publisher" />
   </td>
  </tr>
 </xsl:template>
```

This template generates a row for the base HTML table for each `Book` element in the XML document. Each cell in the table contains an `xsl:value-of` element that instructs the transformation to insert the value of the element indicated by the `select` attribute in the cell.

XSLT is powerful and can be used to perform complex transformations, including transformations tha dynamically vary according to passed parameters, and is not limited to converting XML to HTML. To learn the techniques of XSLT, these books are recommended: *XSLT* and the *XSLT Cookbook* , both from O'Reilly.

See Also

XSLT , by Doug Tidwell (O'Reilly); *XSLT Cookbook* , by Sal Mangano (O'Reilly)

Example 21-5. Transforming XML to HTML (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH21TransformingXMLToHTMLVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH21TransformingXMLToHTMLVB"
 Title="Transforming XML To HTML" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Transform XML To HTML (VB)
 </div>
 <div align="center">
  <br />
  <asp:Xml ID="xmlTransform" Runat="server" />
 </div>
</asp:Content>
```

Example 21-6. Transforming XML to HTML code-behind (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH21TransformingXMLToHTMLVB.aspx
 ''' </summary>
 Partial Class CH21TransformingXMLToHTMLVB
  Inherits System.Web.UI.Page
  '''*********************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   'set the names of the XML and XSLT documents used in the
   'transformation

   xmlTransform.DocumentSource = "xml/books.xml"
   xmlTransform.TransformSource = "xml/books.xslt"
  End Sub 'Page_Load
 End Class 'CH21TransformingXMLToHTMLVB
End Namespace
```

Example 21-7. Transforming XML to HTML code-behind (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH21TransformingXMLToHTMLCS.aspx
 /// </summary>
 public partial class CH21TransformingXMLToHTMLCS : System.Web.UI.Page
 {
   ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///
  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   // set the names of the XML and XSLT documents used in the
   // transformation
   xmlTransform.DocumentSource = "xml//books.xml";
   xmlTransform.TransformSource = "xml//books.xslt";
  } // Page_Load
 } // CH21TransformingXMLToHTMLCS
}
```

Example 21-8. XML source used for transformation

```
<Root>
 <Book>
   <BookID>1</BookID>
  <Title>Access Cookbook</Title>
   <ISBN>0-596-00084-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>2</BookID>
  <Title>ASP.NET Cookbook</Title>
   <ISBN>0-596-00378-1</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>3</BookID>
  <Title>Perl Cookbook</Title>
   <ISBN>1-565-92243-3</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>4</BookID>
  <Title>Java Cookbook</Title>
   <ISBN>0-596-00170-3</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>5</BookID>
  <Title>JavaScript Application Cookbook</Title>
   <ISBN>1-565-92577-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>6</BookID>
  <Title>VB .Net Language in a Nutshell</Title>
   <ISBN>0-596-00092-8</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>7</BookID>
  <Title>Programming Visual Basic .Net</Title>
   <ISBN>0-596-00093-6</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>8</BookID>
  <Title>Programming C#</Title>
   <ISBN>0-596-00117-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>9</BookID>
  <Title>.Net Framework Essentials</Title>
   <ISBN>0-596-00165-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
 <Book>
   <BookID>10</BookID>
  <Title>COM and .Net Component Services</Title>
   <ISBN>0-596-00103-7</ISBN>
  <Publisher>O'Reilly</Publisher>
 </Book>
</Root>
```

Example 21-9. XSLT used to transform HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

 <!-- output main table with header row -->
 <xsl:template match="Root">
  <table width="80%" border="1" cellspacing="0" cellpadding="4" align="center">
   <tr class="tableHeader" bgcolor="#000080">
    <td width="50%" align="center" >Title</td>
    <td width="25%" align="center" >ISBN</td>
    <td width="25%" align="center" >Publisher</td>
   </tr>
   <xsl:apply-templates select="Book" />
  </table>
 </xsl:template>

 <!-- output a row in the table for each Book node in the XML document -->
 <xsl:template match="Book" >
  <tr class="tableCellNormal" bgcolor="#FFFFE0">
  <td width="50%"><xsl:value-of select="Title"/></td>
  <td width="25%" align="center"><xsl:value-of select="ISBN" /></td>
  <td width="25%" align="center"><xsl:value-of select="Publisher" /></td>
  </tr>
 </xsl:template>
</xsl:stylesheet>
```

Recipe 21.5. Determining the User's Browser Type

Problem

Your application requires the use of a specific browser and you want to determine if the required browser is being used before allowing a user to access your application.

Solution

Use the properties of the `Request.Browser` object to determine the browser type and version, and take the action required by your application.

In the code-behind class for the page, use the .NET language of your choice to:

1. Use the `Browser` property of the `Request.Browser` object to return a string representing the full browser type, such as `IE` in the case of Internet Explorer.
2. Use the `Version` property of the `Request.Browser` object to return a string that represents the major and minor version of the browser, such as `6.0` in the case of IE 6.0.
3. Take action accordingly, such as outputting a message indicating whether the user's browser is compatible with the application.

Examples 21-10, 21-11 through 21-12 show the *.aspx* file and the VB and C# code-behind files for an application that demonstrates the solution. The output of our example program is shown inFigure 21-2.

Figure 21-2. Determining the user's browser

Discussion

With different browsers in use today and the significant variation in their capabilities, determining the type of browser being used is common. Based on the browser, you may need to inform the user that the browser is incompatible with your application or output different HTML to support the specific browser.

Our example demonstrates the functionality provided in ASP.NET to determine thebrowser type and version, and then outputs this information along with a message indicating if the user's browser is compatible with the application. The *.aspx* file contains two `asp:Literal` controls used to output messages to the user. The first outputs the browser version information, and the second informs the user if his browser is compatible with the application.

In the `Page_Load` method of the code-behind, the `Browser` and `Version` properties of the `Request.Browser` object are used to create a message to inform the user of the detected browser version. The `Browser` property returns a string such as `IE, Firefox`, or `Opera`. The `Version` property returns a string that represents the major and minor version of the browser. For Internet Explorer 6.0, for instance, the `Browser` property will return `IE`, and the `Version` property will return `6.0`.

Next, the browser type and major version number are compared to the minimum requirements for the application (IE and Version 5). If the browser is IE 5.0 or later, a message is displayed indicating the browser is compatible with the application. Otherwise, a message is displayed indicating the application requires IE 5.0 or later.

For your application, you might want to check the browser version on the home page. If the browser is compatible with your application, output the home page normally. Otherwise, redirect the user to a message page indicating the browser requirements.

The `Request.Browser` object contains many properties that are not used in this example but may be useful in your application. Table 21-1 lists some of the commonly used properties. For a complete list of the available properties, refer to the documentation on the`HttpBrowserCapabilities` class in the MSDN Library.

Table 21-1. Commonly used browser object properties

Property	Description
`Browser`	Returns a string indicating the browser type (`IE, Firefox, Opera`, etc.)
`Cookies`	Returns a Boolean value indicating if the browser supports cookies
`JavaScript`	Returns a Boolean value indicating if the browser supports JavaScript
`MajorVersion`	Returns an integer value indicating the major version of the browser (integer portion of the browser version)
`MinorVersion`	Returns a double value indicating the minor version of the browser (decimal portion of the browser version)
`Version`	Returns a string representing the full browser version (integer and decimal portion)

Properties that return Boolean values, such as `Cookies` and `JavaScript`, indicate what the browser is capable of supporting but not necessarily the current configuration. If the browser supports cookies but the user has configured the browser to disable cookies, the `Cookies` property will still return `true`.

See Also

`HttpBrowserCapabilities` documentation in the MSDN Library

Example 21-10. Determining the user browser type (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH21DeterminingBrowserVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH21DeterminingBrowserVB"
 Title="Determining Browser" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Determining User Browser (VB)
 </div>
 <br />
 <table width="90%" align="center" border="0">
  <tr>
   <td align="center" class="labelText">
    <asp:Literal ID="litBrowser" Runat="server" />
    <br /><br />
    <asp:Literal ID="litMessage" Runat="server" />
   </td>
  </tr>
 </table>
</asp:Content>
```

Example 21-11. Determining the user browser type code-behind (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 ''' CH21DeterminingBrowserVB.aspx
 ''' </summary>
 Partial  Class  CH21DeterminingBrowserVB
  Inherits System.Web.UI.Page
  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   'output user browser
   litBrowser.Text = "Your browser is " & _
    Request.Browser.Browser & " " & _
    Request.Browser.Version

   'check to see if it is an acceptable version
   If ((Request.Browser.Browser.Equals("IE")) AndAlso _
    (Request.Browser.MajorVersion >= 5)) Then
     'output message indicating it is OK
     litMessage.Text = "It is compatible with this application."
   Else
     'output message indicating IE 5 or later must be used
     litMessage.Text = "This application requires IE 5.0 or later."
   End If
  End Sub 'Page_Load
 End Class 'CH21DeterminingBrowserVB
End Namespace
```

Example 21-12. Determining the user browser type code-behind (.cs)

```
using System;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH21DeterminingBrowserCS.aspx
 /// </summary>
 public partial class CH21DeterminingBrowserCS : System.Web.UI.Page
 {
   ///*********************************************************************
  /// <summary>
  /// This routine provides the event handler for the page load event.
  /// It is responsible for initializing the controls on the page.
  /// </summary>
  ///

  /// <param name="sender">Set to the sender of the event</param>
  /// <param name="e">Set to the event arguments</param>
  protected void Page_Load(object sender, EventArgs e)
  {
   // output user browser
   litBrowser.Text = "Your browser is " +
      Request.Browser.Browser + " " +
      Request.Browser.Version;

   // check to see if it is an acceptable version
   if ((Request.Browser.Browser.Equals("IE")) &
    (Request.Browser.MajorVersion >= 5))

   {
    // output message indicating it is OK
    litMessage.Text = "It is compatible with this application.";
   }
   else
   {
    // output message indicating IE 5 or later must be used
    litMessage.Text = "This application requires IE 5.0 or later.";
   }
  } // Page_Load
 } // CH21DeterminingBrowserCS
}
```

PREV

Recipe 21.6. Dynamically Creating Browser-Specific Stylesheets

Problem

You need to vary the look and feel of your application pages depending on the platform (Mac or Windows) being used.

Solution

Place an `asp:Literal` control in the head section of the *.aspx* file, and then set the text property of the control to an HTML style element created programmatically in the code-behind. Use the properties of the `Request.Browser` object to determine the platform type and control the generation of the style element.

In the *.aspx* file, place an `asp:Literal` control in the head section.

In the code-behind class for the page, use the .NET language of your choice to:

1. Use the `Platform` property of the `Request.Browser` object to obtain the browser's platform.
2. Check the platform string for the presence of the substring, such as "`mac`", which indicates whether the browser is running on a Mac platform.
3. Based on the platform, programmatically create the HTML style element.
4. Set the `Text` property of the `asp:Literal` control in the head section of the *.aspx* file to the created HTML style elements.

The *.aspx* file used for this example is shown in Example 21-13. The code-behind is shown in Examples 21-14 (VB) and 21-15 (C#).

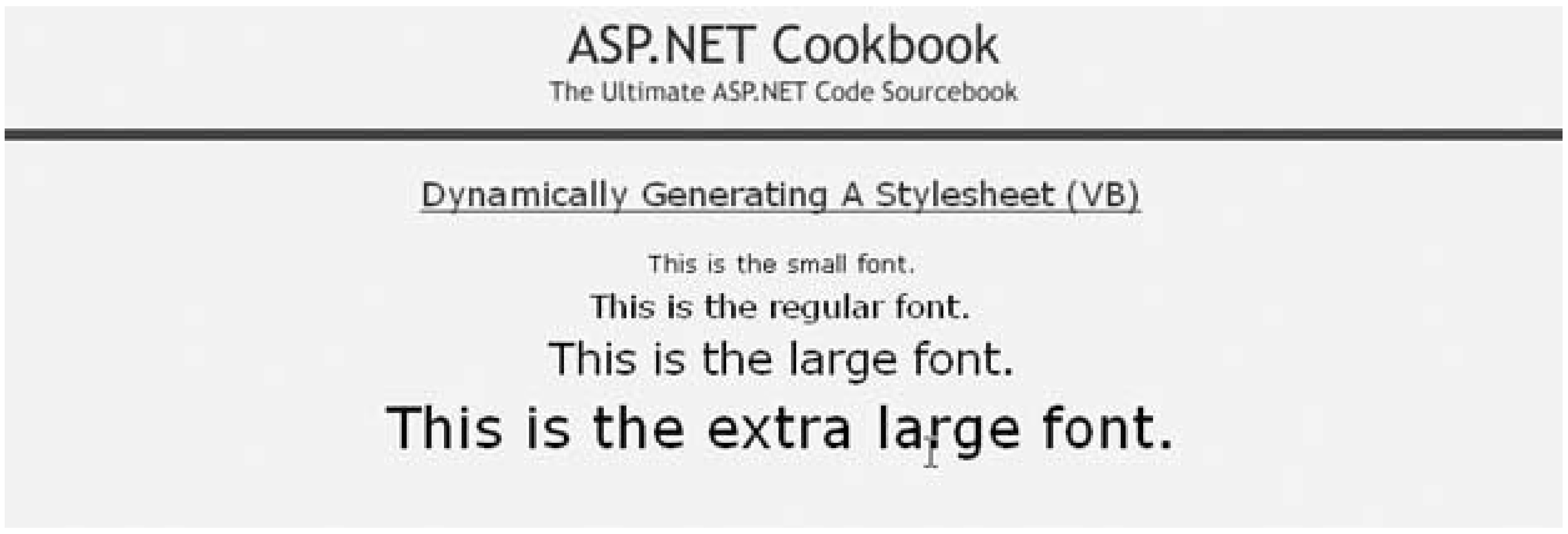
Discussion

HTML is not always rendered the same. Different browsers and platforms render the HTML in various ways. This sometimes requires using a different stylesheet as a function of the browser or platform the browser is running on (Windows, Mac, etc.) to get the same visual effect. Refer to the `Platform` property of the `HttpBrowserCapabilities` class in the MSDN Library for a complete list of platforms detected.

For instance, the displayed size of a font of a given point size is larger on the Windows platform than on the Mac platform. This is caused by the difference in screen resolution. The Mac platform uses a display resolution of 72 dots/inch (DPI), resulting in 1 point being equivalent to 1 pixel. The Windows platform uses a display resolution of 96 DPI, resulting in 1 point being equivalent to 1 1/3 pixels. To display a font the same size on both platforms, the point size must change as a function of the platform.

Our example programmatically generates a different stylesheet depending on whether the browser is running on a Windows or a Mac platform. It creates a stylesheet in the HTML that has four classes defined for a small, regular, large, and extra-large font. This technique is not new to ASP.NET, but the method of generating the stylesheet dynamically is much easier.Figure 21-3 shows the output of our example on a PC platform. Figure 21-4 shows the output on a Mac platform without generating a stylesheet specific to the Mac (the fonts are smaller than the ones inFigure 21-3). Figure 21-5 shows the output on a Mac platform using a specific stylesheet (the fonts are the same size as the ones in Figure 21-3).

Figure 21-3. Example program output for PC platform

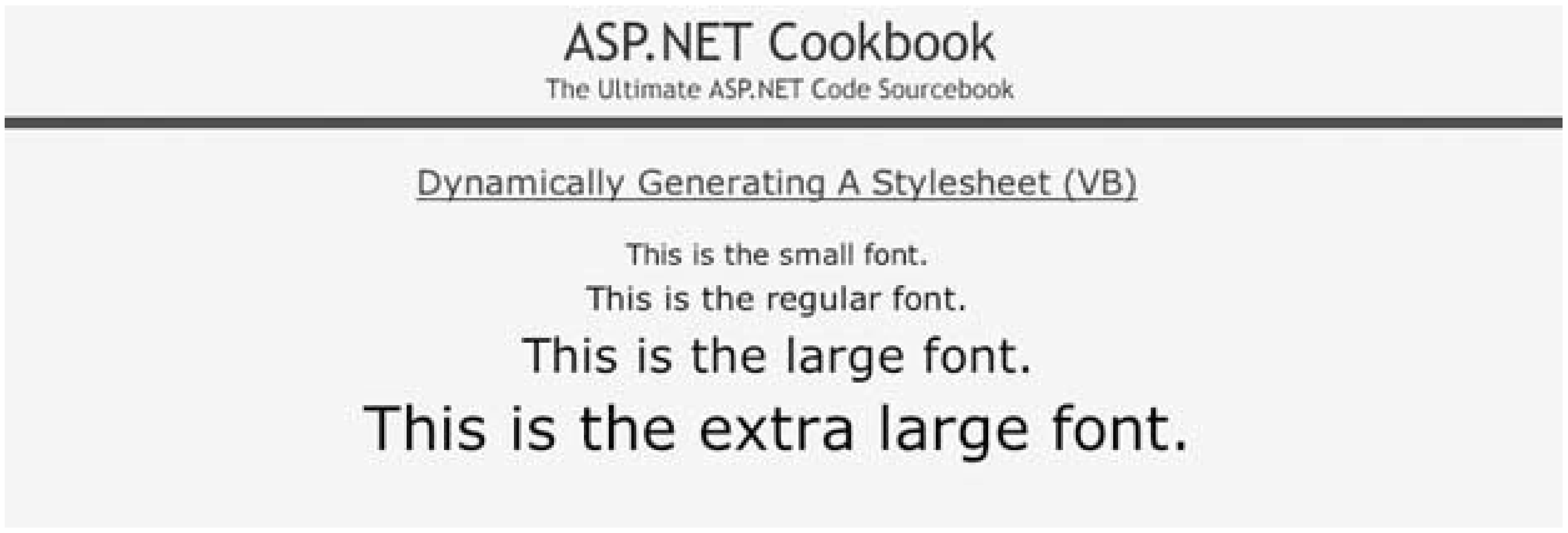

ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Dynamically Generating A Stylesheet (VB)

This is the small font.
This is the regular font.
This is the large font.
This is the extra large font.

Figure 21-4. Example program output for Mac platform without specific stylesheet

Figure 21-5. Example program output for Mac platform with specific stylesheet


ASP.NET Cookbook
The Ultimate ASP.NET Code Sourcebook

Dynamically Generating A Stylesheet (VB)

This is the small font.
This is the regular font.
This is the large font.
This is the extra large font.

Server controls can be placed almost anywhere in an *.aspx* file and are not restricted to the body section of the page. In our example, an `asp:Literal` control is placed in the head section of the *.aspx* file to provide a mechanism to output a stylesheet in the HTML sent to the browser.

```
<head>
 <title>Dynamically Generating Stylesheet</title>
 <link rel="Stylesheet" href="css/ASPNetCookbook.css" />
 <asp:Literal id="litStylesheet" runat="server" />
</head>
```

Any control that returns data when a form is submitted must be within the `open` and `close` form elements.

In the `Page_Load` method of the code-behind, the platform the browser is running on is obtained from the `Platform` property of the `Request.Browser` object. The platform name is converted to lowercase to simplify the check for the specific platform.

The platform string is checked for the presence of the substring `mac`, which indicates the browser is running on a Mac platform.

After determining the platform, four variables are set to indicate the point size to use for the small, regular, large, and extra-large fonts.

Next, a `StringBuilder` is used to create the HTML style element. The style element includes a class for `smallFont, regFont, largeFont`, and `xLargeFont`. Each class contains a font-family style along with a font-size style. The font size is set using the variables described earlier.

Finally, the `Text` property of the `asp:Literal` control placed in the head section of the *.aspx* file is set to the style element.

Here is the resulting style element rendered in the HTML for the Windows platform:

```
<style type='text/css'>
 .smallFont
   {font-family: Verdana, Arial, Helvetica, sans-serif; font-size:8pt;}
 .regFont
   {font-family: Verdana, Arial, Helvetica, sans-serif; font-size:10pt;}
 .largeFont
   {font-family: Verdana, Arial, Helvetica, sans-serif; font-size:14pt;}
 .xLargeFont
   {font-family: Verdana, Arial, Helvetica, sans-serif; font-size:18pt;}
</style>
```

Here is the resulting style element rendered in HTML for the Mac platform:

```
<style type='text/css'>
  .smallFont
    {font-family: Verdana, Arial, Helvetica, sans-serif; font-size:11pt;}
  .regFont
    {font-family: Verdana, Arial, Helvetica, sans-serif; font-size:13pt;}
  .largeFont
    {font-family: Verdana, Arial, Helvetica, sans-serif; font-size:19pt;}
  .xLargeFont
    {font-family: Verdana, Arial, Helvetica, sans-serif; font-size:24pt;}
</style>
```

The classes in the stylesheet can be used like any other class that is hardcoded in the HTML or provided in a cascading stylesheet (CSS):

```
<table width="90%" align="center" border="0">
 <tr>
  <td align="center" class="smallFont">
   This is the small font.
  </td>
</tr>
<tr>
 <td align="center" class="regFont">
  This is the regular font.
 </td>
</tr>
<tr>
 <td align="center" class="largeFont">
```

```
    This is the large font.
   </td>

   </tr>
   <tr>
    <td align="center" class="xLargeFont">
     This is the extra large font.
    </td>
   </tr>
  </table>
```

The solution provided in this recipe can be placed in a user control, giving you the ability to reuse the code in all pages of your application. Refer to Chapter 5 for examples of user controls.

An alternate solution to generating the stylesheet programmatically would be to place a link element in the head section and set the href attribute to a different prebuilt cascading stylesheet as a function of the browser and/or platform. This approach will yield better performance because the stylesheet would not be built for each page request.

To implement the alternate solution, place the following link element in the head section of the *aspx* file. Add the / at the end to close the element or an exception will be thrown when ASP.NET parses the page.

```
<link id="linkCSS" runat="server" rel="stylesheet" />
```

In the Page_Load method of the code-behind, add the href attribute to the linkCSS control setting the value to the required cascading stylesheet. The href attribute must be added using the Add method of the Attributes collection because the generic HTML server control does not have an href property.

```
check the users platform
platform = Request.Browser.Platform.ToLower()

'set font sizes as a function of the platform
If (platform.IndexOf("mac") > -1) Then
 'platform is a Mac so add Mac CSS
 linkCSS.Attributes.Add("href", _
      "css/Mac.css")
Else
 'since not a Mac, assume Windows and add Windows CSS
 '(production app may want to do additional checks if
 'required for styles)
 linkCSS.Attributes.Add("href", _
      "css/Windows.css")
End If
```

```
string platform = null;

// check the users platform
platform = Request.Browser.Platform.ToLower();

// set font sizes as a function of the platform
if (platform.IndexOf("mac") > -1)
{
```

```
  // platform is a Mac so add Mac CSS
  linkCSS.Attributes.Add("href",
       "css/Mac.css");
}
else
{
  // since not a Mac, assume Windows and add Windows CSS
  // (production app may want to do additional checks if
  // required for styles)
  linkCSS.Attributes.Add("href",
       "css/Windows.css");
}
```

The resulting `link` element for the Windows platform is shown next. An `href` attribute has been added to the rendered HTML, with the value set to the required cascading stylesheet:

```
<link id="linkCSS" rel="stylesheet" href="Windows.css"></link>
```

The code for this alternate example can be placed in a user control to provide easy reuse in all of the pages in your application. Refer to Chapter 5 for examples of user controls.

See Also

Chapter 5 for user control examples; MSDN Library for more information on the `HttpBrowserCapabilities` class and `Platform` property values

Example 21-13. Dynamically generated stylesheet (.aspx)

```
<%@ Page Language="VB"
 AutoEventWireup="false"
 CodeFile="CH21DynamicallyGeneratingStyleSheetVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH21DynamicallyGeneratingStyleSheetVB"   %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html  xmlns="http://www.w3.org/1999/xhtml"  >
<head>
 <title>Dynamically Generating Stylesheet</title>
 <link rel="Stylesheet" href="css/ASPNetCookbook.css" />
 <asp:Literal id="litStylesheet" runat="server" />
</head>
<body>
 <form  id="form1"  runat="server">
  <div align="center" class="header">
   <img src="images/ASPNETCookbookHeading_blue.gif" alt="book"/>
  </div>
  <div align="center" class="pageHeading">
   Dynamically Generating A Stylesheet (VB)
  </div>
  <br />
   <table  width="90%"  align="center"  border="0">
   <tr>

    <td align="center" class="smallFont">
    This is the small font.
    </td>
   </tr>
   <tr>
    <td align="center" class="regFont">
    This is the regular font.
    </td>
   </tr>
   <tr>
    <td align="center" class="largeFont">
    This is the large font.
    </td>
   </tr>
   <tr>
    <td align="center" class="xLargeFont">
    This is the extra large font.
    </td>
   </tr>
  </table>
 </form>
</body>
</html>
```

Example 21-14. Dynamically generated stylesheet code-behind (.vb)

```
Option Explicit On
Option Strict On

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 '''  CH21DynamicallyGeneratingStyleSheetVB.aspx
 ''' </summary>
 Partial  Class  CH21DynamicallyGeneratingStyleSheetVB
  Inherits System.Web.UI.Page
  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
   Const FONT_FAMILY As String = _
    "font-family: Verdana, Arial, Helvetica, sans-serif;"

   Dim platform As String
   Dim smFontSize As String
   Dim regFontSize As String
   Dim largeFontSize As String

  Dim xLargeFontSize As String
  Dim styleTag As StringBuilder

  'check the users platform
  platform = Request.Browser.Platform.ToLower()

  'set font sizes as a function of the platform
  If (platform.IndexOf("mac") > -1) Then
   'platform is a Mac
   smFontSize = "11"
   regFontSize = "13"
   largeFontSize = "19"
   xLargeFontSize = "24"
  Else
   'since not a Mac, assume Windows (production app may want to
   'do additional checks if required for styles)
   smFontSize = "8"
   regFontSize = "10"
   largeFontSize = "14"
   xLargeFontSize = "18"
```

```
    End If

    'create style tag
    styleTag = New StringBuilder("<style type='text/css'>" & _
        Environment.NewLine)

    'output the smallFont class
    styleTag.Append(".smallFont {" & FONT_FAMILY & _
      " font-size:" & smFontSize & "pt;}" & _
      Environment.NewLine)

    'output the regFont class
    styleTag.Append(".regFont {" & FONT_FAMILY & _
          " font-size:" & regFontSize & "pt;}" & _
          Environment.NewLine)

    'output the largeFont class
    styleTag.Append(".largeFont {" & FONT_FAMILY & _
         " font-size:" & largeFontSize & "pt;}" & _
         Environment.NewLine)

    'output the xLargeFont class
    styleTag.Append(".xLargeFont {" & FONT_FAMILY & _
          " font-size:" & xLargeFontSize & "pt;}" & _
          Environment.NewLine)

    'close the style tag
    styleTag.Append("</style>" & Environment.NewLine)

    'set literal in Head section to output style sheet
   litStylesheet.Text = styleTag.ToString()
  End Sub 'Page_Load

   End  Class  'CH21DynamicallyGeneratingStyleSheetVB
  End Namespace
```

Example 21-15. Dynamically generated stylesheet code-behind (.cs)

```
using System;
using System.Text;

namespace ASPNetCookbook.CSExamples
{
 /// <summary>
 /// This class provides the code-behind for
 ///  CH21DynamicallyGeneratingStyleSheetCS.aspx
 /// </summary>
```

```
public partial class CH21DynamicallyGeneratingStyleSheetCS :
 System.Web.UI.Page
{
 ///***********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
  const string FONT_FAMILY =
     "font-family: Verdana, Arial, Helvetica, sans-serif;";

  string platform = null;
  string smFontSize = null;
  string regFontSize = null;
  string largeFontSize = null;
  string xLargeFontSize = null;
  StringBuilder styleTag = null;

  // check the users platform
  platform = Request.Browser.Platform.ToLower();

  // set font sizes as a function of the platform
  if (platform.IndexOf("mac") > -1)
  {
   // platform is a Mac
   smFontSize = "11";
   regFontSize = "13";
   largeFontSize = "19";
   xLargeFontSize = "24";
  }
  else
  {

   // since not a Mac, assume Windows (production app may want to
   // do additional checks if required for styles)
   smFontSize = "8";
   regFontSize = "10";
   largeFontSize = "14";
   xLargeFontSize = "18";
  }

  // create style tag
  styleTag = new StringBuilder("<style type='text/css'>" +
      Environment.NewLine);

  // output the smallFont class
  styleTag.Append(".smallFont {" + FONT_FAMILY +
    " font-size:" + smFontSize + "pt;}" +
    Environment.NewLine);

   // output the regFont class
   styleTag.Append(".regFont {" + FONT_FAMILY +
    " font-size:" + regFontSize + "pt;}" +
    Environment.NewLine);

   // output the largeFont class
   styleTag.Append(".largeFont {" + FONT_FAMILY +
    " font-size:" + largeFontSize + "pt;}" +
    Environment.NewLine);

   // output the xLargeFont class
   styleTag.Append(".xLargeFont {" + FONT_FAMILY +
    " font-size:" + xLargeFontSize + "pt;}" +
    Environment.NewLine);

   // close the style tag
   styleTag.Append("</style>" + Environment.NewLine);

   // set literal in Head section to output style sheet
   litStylesheet.Text = styleTag.ToString();
  } // Page_Load
 }  //  CH21DynamicallyGeneratingStyleSheetCS
}
```

Recipe 21.7. Saving and Reusing HTML Output

Problem

To improve the performance of pages that rarely change, you want to capture the output of those pages and save it for reuse when those pages are requested.

Solution

Create the page that contains the desired content as you would any other page, including the server controls you need. At the end of the `Page_Load` method, use the `RenderControl` method of the `Page` control to generate the HTML, and save the HTML to a file.

In the code-behind class for the page, use the .NET language of your choice to:

1. Create an `HtmlTextWriter` to use for rendering the page.
2. Use the `RenderControl` method of the `Page` control to render the output of the page to the `HtmlTextWriter`.
3. Save the rendered output to a file and redirect to another page.

Examples 21-16 and 21-17 show the VB and C# code-behind files for our application that demonstrates this solution.

Discussion

Occasionally, it's beneficial to save the HTML output from a generated page. This is commonly done when using the saved HTML can significantly improve web site performance. If the content of a page is static, for example, there is no point in dynamically generating HTML each time the page is requested. Until the advent of ASP.NET, the only way to save the HTML was to use the "Save as Complete Web Page" feature of Internet Explorer or another browser. Though this method does save the HTML, it copies all of the page images to the local machine and changes the image references to point to the local copies. If you are trying to improve performance by capturing a static copy of the page to use on your web server, this technique will work poorly.

With ASP.NET, you can easily capture the HTML exactly as it would be sent to the browser. For our example that illustrates this solution, we have used the page from Recipe 21.3 and added code to th
`Page_Load` method to save the rendered output.

The `RenderControl` method of the `Page` control provides the ability to render the output of the page tc the HtmlTextWriter passed to the method. Unfortunately, the HtmlTextWriter does not provide any methods for reading the contents, so a little more work is required to access the rendered HTML.

By creating a StringBuilder and then using it to create a StringWriter, which is used to create the required HtmlTextWriter, the contents of the HtmlTextWriter are available by way of the original StringBuilder. This works because the underlying storage mechanism for the StringWriter is a StringBuilder, and because the StringWriter (a stream) is used to create the HtmlTextWriter, the RenderControl method is writing the rendered output to the StringBuilder. Here is our example code to accomplish this:

VB

```
renderedOutput = New StringBuilder()
strWriter = New StringWriter(renderedOutput)
tWriter = New HtmlTextWriter(strWriter)
```

C#

```
renderedOutput = new StringBuilder();
strWriter = new StringWriter(renderedOutput);
tWriter = new HtmlTextWriter(strWriter);
```

After creating the HtmlTextWriter, the RenderControl method of the Page is called to render the HTML for the page:

```
Page.RenderControl(tWriter)
```

```
Page.RenderControl(tWriter);
```

Now that the rendered HTML is available, it needs to be saved to a file on the server. This can be accomplished by creating the file with a FileStream and using a StreamWriter to write the rendered output in the StringBuilder to the file:

```
filename = Server.MapPath(".") & "\" & OUTPUT_FILENAME
outputStream = New FileStream(filename, _
      FileMode.Create)
sWriter = New StreamWriter(outputStream)
sWriter.Write(renderedOutput.ToString())
sWriter.Flush()
```

```
filename = Server.MapPath(".") + "\\" + OUTPUT_FILENAME;
outputStream = new FileStream(filename,
      FileMode.Create);
```

```
 sWriter = new StreamWriter(outputStream);
 sWriter.Write(renderedOutput.ToString());
 sWriter.Flush();
```

The last step is to redirect to another page. This is necessary because allowing the page to be displayed would result in an additional rendering and an exception being thrown indicating the page has more than one server-side form element. If you need the page to be displayable anyway, a parameter can be passed in the `querystring` and checked in the code to determine if the output should be rendered and written to a file or handled normally.

This technique can be used for individual controls in the same manner as for the entire page. For example, if you have a page that contains a `DataGrid` and you want the rendered HTML for just the `DataGrid`, you can call the `RenderControl` method of the `DataGrid` and then save the output, as described earlier.

See Also

Recipe 21.3

Example 21-16. Capturing rendered output (.vb)

```
Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load

 Const OUTPUT_FILENAME As String = "CH21CaptureRenderedOutputVB.html"

 Dim renderedOutput As StringBuilder
 Dim strWriter As StringWriter

 Dim tWriter As HtmlTextWriter
 Dim outputStream As FileStream = Nothing
 Dim sWriter As StreamWriter = Nothing
 Dim filename As String
 Dim nextPage As String

 Try
  'set the names of the XML and XSLT documents used in the
  'transformation
  xmlTransform.DocumentSource = "xml/books.xml"
  xmlTransform.TransformSource = "xml/books.xslt"

  'create a HtmlTextWriter to use for rendering the page
  renderedOutput = New StringBuilder
  strWriter = New StringWriter(renderedOutput)
  tWriter = New HtmlTextWriter(strWriter)
```

```
    'render the page output
    Page.RenderControl(tWriter)

    'save the rendered output to a file
    filename = Server.MapPath(".") & "\" & OUTPUT_FILENAME
    outputStream = New FileStream(filename, _
          FileMode.Create)
    sWriter = New StreamWriter(outputStream)
    sWriter.Write(renderedOutput.ToString())
    sWriter.Flush()

    'redirect to another page
    'NOTE: Continuing with the display of this page will result in the
    '  page being rendered a second time which will cause an exception
    '  to be thrown
    nextPage = "DisplayMessage.aspx?" & _
         "PageHeader=Information" & "&" & _
         "Message1=HTML Output Saved To " & OUTPUT_FILENAME
    Response.Redirect(nextPage)

  Finally
    'clean up
    If (Not IsNothing(outputStream)) Then
     outputStream.Close()
    End If

    If (Not IsNothing(tWriter)) Then
     tWriter.Close()
    End If

    If (Not IsNothing(strWriter)) Then
     strWriter.Close()
    End If
  End Try
End Sub 'Page_Load
```

Example 21-17. Capturing rendered output (.cs)

```
protected void Page_Load(object sender, EventArgs e)
{
  const  string  OUTPUT_FILENAME  =  "CH21CaptureRenderedOutputCS.html";

  StringBuilder renderedOutput = null;
  StringWriter strWriter = null;
  HtmlTextWriter tWriter = null;
  FileStream outputStream = null;
  StreamWriter sWriter = null;
```

```
String filename = null;
String nextPage = null;

try
{
 // set the names of the XML and XSLT documents used in the
 // transformation
 xmlTransform.DocumentSource = "xml//books.xml";
 xmlTransform.TransformSource = "xml//books.xslt";

 // create a HtmlTextWriter to use for rendering the page
 renderedOutput = new StringBuilder();
 strWriter = new StringWriter(renderedOutput);
 tWriter = new HtmlTextWriter(strWriter);

 // render the page output
 Page.RenderControl(tWriter);

 // save the rendered output to a file
 filename = Server.MapPath(".") + "\\" + OUTPUT_FILENAME;
 outputStream = new FileStream(filename,
       FileMode.Create);
 sWriter = new StreamWriter(outputStream);
 sWriter.Write(renderedOutput.ToString());
 sWriter.Flush();

 // redirect to another page
 // NOTE: Continuing with the display of this page will result in the
 //       page being rendered a second time which will cause an exception
 //       to be thrown
 nextPage =  "DisplayMessage.aspx?" +
   "PageHeader=Information" + "&" +
   "Message1=HTML Output Saved To " + OUTPUT_FILENAME;
 Response.Redirect(nextPage);
}

finally
{

 // clean up
 if (outputStream != null)
 {
  outputStream.Close();
 }

 if (tWriter != null)
 {
  tWriter.Close();
 }

 if (strWriter != null)
 {
    strWriter.Close();
  }
 }
} // Page_Load
```

PREV

Recipe 21.8. Sending Mail

Problem

You want the user of your application to be able to send email.

Solution

Create a form to allow the user to enter the email information. When the user submits the form to the server, build a `MailMessage` object from the email information and then send the email using the `SmtpClient` class.

In the *.aspx* file:

1. Create a form to capture the sender's email address, the recipient's email address, the subject, and the message.
2. Add a Send button that initiates the sending of the email.

In the code-behind class for the page, use the .NET language of your choice to:

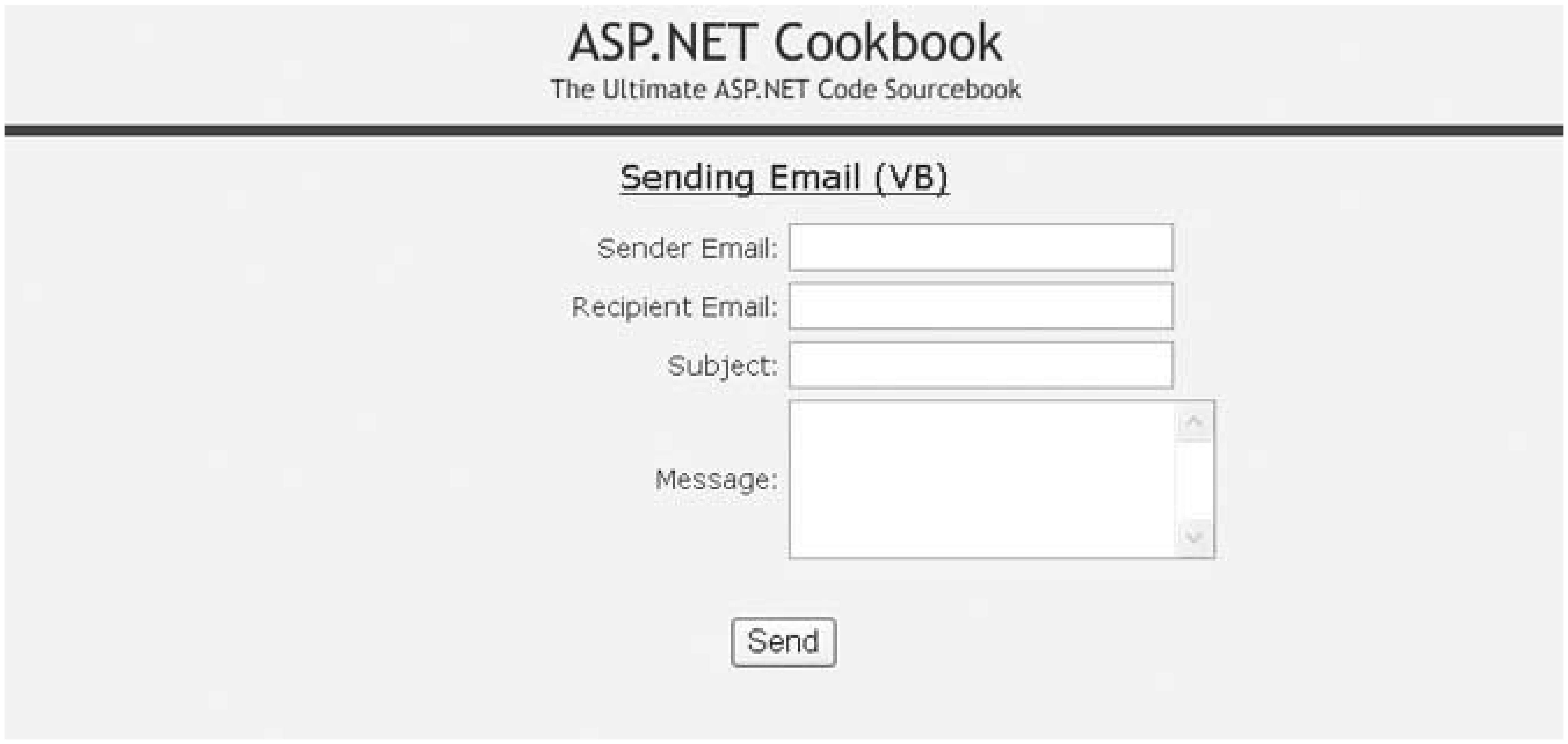
1. Create a `MailMessage` object, which is used as a container for the mail message, setting the sender email address, recipient email address, subject, and body using the constructor parameters.
2. Create a `SmtpClient` object setting the email server using the constructor parameter.
3. Call the `Send` method to perform the send operation.

Examples 21-18 , 21-19 through 21-20 show the *.aspx* file and the VB and C# code-behind files for an application we've written to demonstrate this solution. Theoutput of the application is shown in Figure 21-6 .

Discussion

Sending email is a common requirement in web applications. In classic ASP, third-party controls are required to send email. In ASP.NET, all of the functionality required to send email is provided and is very easy to use.

Figure 21-6. Send email form output



To send email, you need the sender's email address, the recipient's email address, the subject, and the message. In our example that illustrates this solution, a form is used to collect the information. The form includes a Send button that initiates the sending of the email.

When the Send button is clicked, the `btnSend_ServerClick` method in the code-behind is executed. This method is responsible for collecting the information from the form and sending the email. For simplicity, no validation is performed on the data in our example; however, you should provide validation of the data in your application. Refer to Chapter 3 for data validation examples.

The first step in sending an email is to create a `MailMessage` object. This object is used as a container for the mail message, as shown in our example. The sender email address, recipient email address, subject, and message body are set using the constructor parameters.

The sender email address, recipient email address, and subject should always be set to valid values. Spam filters typically check these fields, and, if any are blank, the mail message may be branded as spam. In addition, some spam filters check the format of the recipient email address, and the more thorough spam filters will verify if the sender email address is valid.

After the `MailMessage` is initialized, a `SmtpClient` object is created setting the email server that will send the email message using the constructor parameter and the `Send` method will be called to perform the send operation.

In Version 2.0 of the .NET Framework, a new namespace (`System.Net.Mail`) has been added to provide support for sending email. The classes provided in `System.Web.Mail` in Version 1.x have been deprecated and should not be used for new applications. They are still provided to support applications developed for 1.x, though.

Our example shows the case of an email sent to a single recipient. To send the email to multiple recipients, add the recipient email addresses to the To collection property.

Copies and blind copies can be sent by adding the email address of the copied recipients to the `CC` collection property and the email address of the blind copy recipient to the `Bcc` collection property.

Attachments can be included with the email by adding `MailAttachment` objects to the `Attachments` collection, as shown here, where [*`filename`*] is the fully qualified path to the file that you need to attach to the mail message:

VB

```
emailMessage.Attachments.Add(New Attachment([filename]))
```

C#

```
emailMessage.Attachments.Add(new Attachment([filename]));
```

See Also

Chapter 3 for data validation

Example 21-18. Sending email (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/ASPNetCookbookVB.master"
 AutoEventWireup="false"
 CodeFile="CH21SendingEmailVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH21SendingEmailVB"
 Title="Sending Email" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Sending Email (VB)
 </div>
 <table width="60%" align="center" border="0">
  <tr>
   <td width="50%" align="right" class="labelText">Sender Email: </td>
   <td width="50%">
    <input id="txtSenderEmail" runat="server" />
   </td>
  </tr>
  <tr>
   <td width="50%" align="right" class="labelText">Recipient Email: </td>
   <td width="50%">
    <input id="txtRecipientEmail" runat="server" />
   </td>
  </tr>
  <tr>
   <td width="50%" align="right" class="labelText">Subject: </td>
```

```
    <td width="50%">
     <input id="txtSubject" runat="server" />
    </td>
   </tr>
   <tr>
    <td width="50%" align="right" class="labelText">Message: </td>
    <td width="50%">
     <textarea id="txtMessage" runat="server" rows="4" cols="20"></textarea>
    </td>
   </tr>
   <tr>
    <td colspan="2" align="center">
     <br />
     <input id="btnSend" runat="server"
     type="button"
     value="Send"
     onserverclick="btnSend_ServerClick" />
    </td>
   </tr>
  </table>
</asp:Content>
```

Example 21-19. Sending email code-behind (.vb)

```
Option Explicit On
Option Strict On

Imports System
Imports System.Net.Mail

Namespace ASPNetCookbook.VBExamples
 ''' <summary>
 ''' This class provides the code-behind for
 ''' CH21SendingEmailVB.aspx
 ''' </summary>
 Partial Class CH21SendingEmailVB
  Inherits System.Web.UI.Page
  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the page load event. It
  ''' is responsible for initializing the controls on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Private Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Me.Load
  End Sub 'Page_Load

  '''*************************************************************************
  ''' <summary>
  ''' This routine provides the event handler for the send button click

  ''' event. It is responsible for sending an email based on the data
  ''' entered on the page.
  ''' </summary>
  '''
  ''' <param name="sender">Set to the sender of the event</param>
  ''' <param name="e">Set to the event arguments</param>
  Protected Sub btnSend_ServerClick(ByVal sender As Object, _
          ByVal e As System.EventArgs)
   Dim emailMessage As MailMessage
   Dim emailAttachement As Attachment
   Dim filename As String
   Dim client As SmtpClient
   Dim emailServer As String

   'build the mail message with an attachment
   emailMessage = New MailMessage(txtSenderEmail.Value, _
          txtRecipientEmail.Value, _
          txtSubject.Value, _
          txtMessage.Value)

   'add an attachment
   filename = Server.MapPath("downloads") & "\SampleEmailAttachment.txt"
   emailAttachement = New Attachment(filename)
   emailMessage.Attachments.Add(emailAttachement)

   'get the email server and send the email
   emailServer = ConfigurationManager.AppSettings("EmailServer")
   client = New SmtpClient(emailServer)
   client.Send(emailMessage)
  End Sub 'btnSend_ServerClick
 End Class 'CH21SendingEmailVB
End Namespace
```

Example 21-20. Sending email code-behind (.cs)

```
using System;
using System.Configuration;
using System.Net.Mail;

namespace ASPNetCookbook.CSExamples
{
```

```
/// <summary>
/// This class provides the code-behind for
///  CH21SendingEmailCS.aspx
/// </summary>
public partial class CH21SendingEmailCS : System.Web.UI.Page
{
  ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the page load event.
 /// It is responsible for initializing the controls on the page.
 /// </summary>
 ///

 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void Page_Load(object sender, EventArgs e)
 {
 } // Page_Load

  ///*********************************************************************
 /// <summary>
 /// This routine provides the event handler for the send button click
 /// event. It is responsible for sending an email based on the data
 /// entered on the page.
 /// </summary>
 ///
 /// <param name="sender">Set to the sender of the event</param>
 /// <param name="e">Set to the event arguments</param>
 protected void btnSend_ServerClick(Object sender,
           System.EventArgs e)
 {
  MailMessage emailMessage;
  Attachment emailAttachement;
  String filename;
  SmtpClient client;
  String emailServer;

  // build the mail message with an attachment
  emailMessage = new MailMessage(txtSenderEmail.Value,
         txtRecipientEmail.Value,
         txtSubject.Value,
         txtMessage.Value);

  // add an attachment
  filename = Server.MapPath("downloads") + "\\SampleEmailAttachment.txt";
  emailAttachement = new Attachment(filename);
  emailMessage.Attachments.Add(emailAttachement);

  // get the email server and send the email
  emailServer = ConfigurationManager.AppSettings["EmailServer"];
  client = new SmtpClient(emailServer);
  client.Send(emailMessage);
```

```
   } //btnSend_ServerClick
  } // CH21SendingEmailCS
}
```

PREV

Recipe 21.9. Dynamic Menus

Problem

You want to use a dynamic menu in your application to provide a compact means for a user to navigate within your application, and you want to accomplish this without using a third-party control.

Solution

Create a *Web.sitemap* file to define the pages and navigation in your application. Create a *.master* file containing a `SiteMapDataSource` control and a `Menu` control with the `DataSourceID` set to the `ID` of the `SiteMapDataSource` control and other HTML common to all pages in your application. Use the *.master* file as the master page for all pages in your application that require the dynamic menu.

In the *Web.sitemap* file:

1. Add a `siteMapNode` element for each section of your application.
2. Add a `siteMapNode` element for each page in the sections.

In the *.master* file:

1. Add a `SiteMapDataSource` control.
2. Add a `Menu` control.
3. Set the `DataSourceID` attribute of the `Menu` control to the value of the `ID` attribute of the `SiteMapDataSource` control.

In the *.aspx* files that require the dynamic menu, set the `MasterPageFile` attribute of the `@ Page` directive to the name of the *.master* file.

Example 21-21 shows the *Web.sitemap* file, Example 21-22 shows the *.master* file, and Example 21-23 shows the *.aspx* file for this example. Figure 21-7shows the output of the test page with the dynamic menu in the static state. Figure 21-8shows the output with the dynamic menu expanded.

Figure 21-7. Dynamic menu in static state

Discussion

Dynamic menus are common in applications. They provide an excellent means of navigating complex applications while minimizing the page real estate that must be dedicated to the menu. In ASP.NET 1.x, no support for dynamic menus was provided. You had to use a third-party control or take on the daunting task of creating your own using JavaScript and DHTML. ASP.NET 2.0 provides a site navigation infrastructure that can be used to create dynamic menus, as well as breadcrumb trails (see Recipe 21.9), without writing any code.

Figure 21-8. Dynamic menu expanded

The first step to creating a dynamic menu is to create a *Web.sitemap* file. The *Web.sitemap* file is an XML file that provides the definition of the page structure of your application and the navigation to the pages within your application. The file consists of a hierarchical structure of `siteMapNode` elements that define the levels for the dynamic menu:

```
 <?xml version="1.0" encoding="utf-8" ?>
 <siteMap  xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0"  >
  <siteMapNode>
   <siteMapNode title="Online Examples"
    description="Run the examples on-line">
     <siteMapNode title="1. Master Pages"
     description="Master Page Recipes">
       <siteMapNode url="CH01QuickContentPageVB.aspx"
     title="Quick Master Page"
     description="Quick Master Page" />
       <siteMapNode url="CH01NestedMasterContentPageVB.aspx"
     title="Nested Master Pages"
     description="Nested Master Pages" />
       <siteMapNode url="CH01SetMasterPageAtRuntimeVB.aspx"
     title="Set Master Page At Runtime"
     description="Set Master Page At Runtime" />
   </siteMapNode>

  …

 </siteMapNode>
</siteMap>
```

The `url` attribute defines the URL of the page that will be displayed when the menu item is selected. The `title` attribute provides the text that will be displayed in the menu and the`description` attribute provides the text for the tool tip that will be displayed when the user hovers over the menu item.

The `url` attribute can be omitted for menu items that are used to group child items but have no page associated with them.

The value of the `url` attribute must be unique for every `siteMapNode` element in the *Web.sitemap* file. If the file contains duplicate `urls` , an exception will be thrown the first time the file is read.

The next step is to create *.master* file with a `SiteMapDataSource` control and a `Menu` control with the `DataSourceID` attribute of the `Menu` control set to the `ID` of the `SiteMapDataSource` control. By default, the `SiteMapDataSource` uses the `XmlSiteMapProvider` supplied with ASP.NET 2.0 to provide a data source that can be used by the `Menu` control to render the dynamic menu.

Dynamic menus are generally used on many pages within an application. To support the reuse, place the menu in a user control or a master page.

The `Menu` control has many attributes that can be used to define the look and feel of the dynamic menu. The menu can be displayed horizontally or vertically by setting the `Orientation` attribute to the desired orientation. The number of menu levels displayed all of the time is controlled by the `StaticDisplayLevels` attribute. Other attributes are available to define nearly every aspect of the menu. Refer to the `Menu` control documentation in the MSDN Library for information on all of the available attributes.

The *Web.sitemap* file requires a single `siteMapNode` element as a child of the `siteMap` root element. To display a menu that has multiple items at the top level, leave all attributes of the first `siteMapNode` element undefined and set the `StaticDisplayLevels` attribute of the `Menu` control to `2` .

The final step is to use the *.master* file you created to display the dynamic menu throughout your application. Refer to the recipes in Chapter 1for more information on using master pages.

The site navigation infrastructure provides an excellent means to define the structure of your application and the navigation within it. By using the *Web.sitemap* file, any changes to the navigation can be made in a single place with no code changes or recompilation required. The site navigation infrastructure can be used to generate the breadcrumb trail used in many applications to display the current location within the application and to provide an easy way to navigate back to a previous location. Refer to Recipe 21.9 for an example of using the infrastructure to display a breadcrumb trai

See Also

Chapter 1 ; Recipe 21.9; the `Menu` control in the MSDN Library

Example 21-21. Web.sitemap file

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
 <siteMapNode>
  <siteMapNode title="Online Examples"
    description="Run the examples on-line">
     <siteMapNode title="1. Master Pages"
    description="Master Page Recipes">
    <siteMapNode url="CH01QuickContentPageVB.aspx"
    title="Quick Master Page"
    description="Quick Master Page" />
    <siteMapNode url="CH01NestedMasterContentPageVB.aspx"
    title="Nested Master Pages"
    description="Nested Master Pages" /
    <siteMapNode url="CH01SetMasterPageAtRuntimeVB.aspx"
    title="Set Master Page At Runtime"
    description="Set Master Page At Runtime" />
  </siteMapNode>
  <siteMapNode title="2. Tabular Data"
    description="Tabular Data Recipes">
    <siteMapNode url="CH02QuickAndDirtyGridViewVB.aspx"
    title="Quick Tabular Display"
    description="Quick Tabular Display" />
    <siteMapNode url="CH02TemplatesWithRepeaterVB.aspx"
    title="Templates With Repeater"
    description="Templates With Repeater" />
    <siteMapNode url="CH02DatagridAscDescSortingVB.aspx"
    title="DataGrid With Sorting"
    description="DataGrid With Sorting" />
    <siteMapNode url="CH02GridViewWithTotalsRowVB.aspx"
    title="GridView With Totals Row"
    description="GridView With Totals Row" />
  </siteMapNode>
  <siteMapNode title="3. Validation"
    description="Validation Recipes">
    <siteMapNode url="CH03RequiredFieldValidationVB.aspx"
    title="Required Field Validation"
    description="Required Field Validation" />
    <siteMapNode url="CH03RangeValidationVB.aspx"
    title="Range Validation"
    description="Range Validation" />
  </siteMapNode>

  <siteMapNode title="4. Forms"
    description="Form Recipes">
     <siteMapNode url="CH04SettingDefaultSubmitButtonCS.aspx"
    title="Default Submit Button"
    description="Default Submit Button" />
     <siteMapNode url="CH04SurveyDataCS1.aspx"
    title="Using Wizard Control"
    description="Using Wizard Control" />
     <siteMapNode url="CH04SetFocusCS1.aspx"
    title="Setting Initial Focus"
    description="Setting Initial Focus" />
  </siteMapNode>
  <siteMapNode title="5. User Controls"
    description="User Control Recipes">
     <siteMapNode url="CH05DisplayHeaderCS.aspx"
    title="Page Header"
    description="Page Header" />
     <siteMapNode url="CH05UserControlCommTestCS.aspx"
    title="User Control Communication"
    description="User Control Communication" />
  </siteMapNode>
  <siteMapNode title="6. Custom Controls"
    description="Custom Control Recipes">
```

```
      <siteMapNode url="CH06DisplayQuickAndDirtyControlCS1.aspx"
       title="Quick and Dirty Custom Control"
       description="Quick and Dirty Custom Control" />
      <siteMapNode url="CH06DisplayControlWithStateCS1.aspx"
       title="Custom Control With State"
       description="Custom Control With State" />
    </siteMapNode>
     <siteMapNode title="7. Maintaining State"
      description="Maintaining State Recipes" />
   <siteMapNode title="8. Error Handling"
     description="Error Handling Recipes" />
   <siteMapNode title="9. Security"
     description="Security Recipes" />
   <siteMapNode title="10. Personalization"
     description="Personalization Recipes" />
   <siteMapNode title="11. Web Parts"
     description="Web Parts Recipes" />
   <siteMapNode title="12. Configuration"
     description="Configuration Recipes" />
   <siteMapNode title="13. Tracing and Debugging"
     description="Tracing and Debugging Recipes" />
   <siteMapNode title="13. Web Services"
     description="Web Services Recipes" />
   <siteMapNode title="14. Web Services"
     description="Web Services Recipes" />
   <siteMapNode title="15. Dynamic Images"
     description="Dynamic Image Recipes" />
   <siteMapNode title="16. Caching"
     description="Caching Recipes" />

   <siteMapNode title="17. Internationalization"
     description="Internationalization Recipes" />
   <siteMapNode title="18. File Operations"
     description="File Operation Recipes" />
   <siteMapNode title="19. Performance"
     description="Performance Recipes" />
   <siteMapNode title="20. Http Handlers"
     description="Http Handler Recipes" />
   <siteMapNode title="21. Assorted Tips"
     description="Assorted Tips Recipes">
     <siteMapNode url="CH21TestDynamicMenuVB.aspx"
     title="Dynamic Menus"
     description="Dynamic Menus" />
     <siteMapNode url="CH21TestBreadcrumbsVB.aspx"
     title="Dynamic Menus with Breadcrumbs"
     description="Dynamic Menus with Breadcrumbs" />
  </siteMapNode>
 </siteMapNode>
 <siteMapNode url="CodeDownload.aspx"
     title="Downloads"
     description="Download the code and database for the book">
   <siteMapNode url="ASPNetCookbook2VB.zip"
```

```
    title="VB Code"
    description="Download the VB.NET Code" />
   <siteMapNode  url="ASPNetCookbook2CS.zip"
    title="C# Code"
    description="Download the C# Code" />
  <siteMapNode url="ASPNetCookbookDB.zip"
    title="Database"
    description="Download the database" />
 </siteMapNode>
 <siteMapNode url="Feedback.aspx"
    title="Feedback"
    description="Send us feedback on the book" />
 <siteMapNode url="Errata.aspx"
    title="Errata"
    description="Submit errata" />
  </siteMapNode>
</siteMap>
```

Example 21-22. Dynamic menu master page (.master)

```
<%@ Master Language="VB"
     AutoEventWireup="false" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html  xmlns="http://www.w3.org/1999/xhtml"  >
<head  id="Head1"  runat="server">
 <title>Dynamic Menu Master Page</title>
 <link rel="Stylesheet" href="css/ASPNetCookbook.css" />
</head>
<body>

 <form  id="form1"  runat="server">
  <div class="headerBlue">
   <div class="headerLeftColumn">
    <img src="images/ASPNETCookbookHeading_blue.gif" alt="book"/>
   </div>
   <div class="headerRightColumn">
    <asp:SiteMapDataSource ID="menuSource" runat="server" />
    <asp:menu ID="Menu1" runat="server"
    DataSourceID="menuSource"
    Orientation="Horizontal"
    StaticDisplayLevels="2"
    BackColor="#f0f0f0"
    ForeColor="#6B0808"
    Font-Size="8pt"
    Font-Bold="true"
    StaticMenuItemStyle-HorizontalPadding="5"
    StaticMenuItemStyle-VerticalPadding="0"
    StaticHoverStyle-BackColor="#6B0808"
    StaticHoverStyle-ForeColor="#FFFFFF"
    DynamicMenuItemStyle-BackColor="#f0f0f0"
    DynamicMenuItemStyle-ForeColor="#0000A0"
    DynamicMenuItemStyle-BorderStyle="Solid"
    DynamicMenuItemStyle-BorderColor="#c0c0c0"
    DynamicMenuItemStyle-HorizontalPadding="5"
    DynamicMenuItemStyle-BorderWidth="1"
    DynamicHoverStyle-BackColor="#6B0808"
    DynamicHoverStyle-ForeColor="#FFFFFF"
    width="100%"
    Height="35" >
    </asp:menu>
   </div>
  </div>
 <div>
  <asp:ContentPlaceHolder ID="PageBody" Runat="server" >
   <div align="center">
    <br />
    <br />
     <h4>Default Content Displayed When No Content Is Provided
     In Content Pages</h4>
   </div>
  </asp:ContentPlaceHolder>
 </div>
  </form>
</body>
</html>
```

Example 21-23. Test page for dynamic menu (.aspx)

```
<%@ Page Language="VB" MasterPageFile="~/CH21DynamicMenu.master"
 AutoEventWireup="false"
 CodeFile="CH21TestDynamicMenuVB.aspx.vb"
 Inherits="ASPNetCookbook.VBExamples.CH21TestDynamicMenuVB"

 title="Test Dynamic Menu" %>
<asp:Content ID="pageBody" runat="server" ContentPlaceHolderID="PageBody">
 <div align="center" class="pageHeading">
  Test Dynamic Menu (VB)
 </div>
</asp:Content>
```

PREV

Recipe 21.10. Adding Breadcrumbs

Problem

You want to use a dynamic menu in your application and display abreadcrumb trail to show the current location within the application and to provide an easy way for the user to navigate back to a previous location.

Solution

Implement the dynamic menu solution described in Recipe 21.8 and add a `SiteMapPath` control to the *.master* file. Use the *.master* file as the master page for all pages in your application that require the dynamic menu and the breadcrumb trail.

Example 21-24 shows the *.master* file used to display the dynamic menu and breadcrumb trail. Figure 21-9 shows the output a sample page with the breadcrumb trail displayed.

Figure 21-9. Sample page with dynamic menu and breadcrumb trail

Discussion

To improve the user experience, applications commonly display a breadcrumb trail to show the current location within the application and to provide an easy way to navigate back to a previous location. The site navigation infrastructure provided in ASP.NET 2.0 can be used to display a breadcrumb trail using the data in the *Web.sitemap* file that is also used to as the data source for dynamic menus (see Recipe 21.8).

Once you have the site structure defined in the *Web.sitemap* file, as described in Recipe 21.8, all that is required is to add a `SiteMapPath` control to the *.master* file that will be used as the master page for your application pages that need to display a breadcrumb trail.

The `SiteMapPath` control uses the site map provider defined by the `SiteMapProvider` attribute. If the `SiteMapProvider` attribute is not defined, the `SiteMapPath` control will use the default `SiteMapProvider`, which by default is the `XmlSiteMapProvider`.

The `SiteMapPath` control displays a series of nodes that match the hierarchy defined in the *Web.sitemap* file from the root `SiteMapNode` to the `SiteMapNode` used to define the currently displayed page. All nodes from the root node to the current page node will be displayed in the breadcrumb trail. By default, all nodes except the current node will be defined as hyperlinks to provide the ability to navigate back up the hierarchy.

If a node in the *Web.sitemap* file does not have the `url` attribute defined, the node text will be displayed in the breadcrumb trail but it will not be a hyperlink.

If you want the current node displayed as a hyperlink, you can set the `RenderCurrentNodeAsLink` attribute to `true`.

If a page is displayed that does not have a `SiteMapNode` element in the *Web.sitemap* file, the breadcrumb trail will not be displayed.

The `SiteMapPath` control provides many attributes that can be used to define the look and feel of the rendered breadcrumb trail. In addition, it provides the ability to define templates to be used for the root node (`RootNodeTemplate`), the current node (`CurrentNodeTemplate`), the nodes in between the root and current node (`NodeTemplate`), and the separator used between the nodes (`PathSeparatorTemplate`). The combination of the templates and attributes provides nearly unlimited ways to display your breadcrumb trail.

The `XmlSiteMapProvider` along with the `SiteMapDataSource`, `Menu`, and `SiteMapPath` controls will meet the menu system needs of most applications. Other controls can be used if they do not meet the needs of your application, such as using the `treeView` control instead of the `Menu` control. In addition, all of the controls can be manipulated programmatically if you need to adjust their output more.

See Also

Recipe 21.8; the `Menu`, `SiteMapPath`, and `treeView` controls in the MSDN Library

Example 21-24. Master page with dynamic menu and breadcrumb trail (.master)

```
<%@ Master Language="VB"
```

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

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
 <title>Dynamic Menu Master Page</title>
 <link rel="Stylesheet" href="css/ASPNetCookbook.css" />
</head>
<body>
 <form id="form1" runat="server">
  <div class="headerBlue">
   <div class="headerLeftColumn">
    <img src="images/ASPNETCookbookHeading_blue.gif" />
   </div>
   <div class="headerRightColumn">
    <asp:SiteMapDataSource ID="menuSource" runat="server" />
     <asp:menu ID="Menu1" runat="server"
    DataSourceID="menuSource"
    Orientation="Horizontal"
     StaticDisplayLevels="2"
     BackColor="#f0f0f0"
     ForeColor="#6B0808"
     Font-Size="8pt"
    Font-Bold="true"
      StaticMenuItemStyle-HorizontalPadding="5"
      StaticMenuItemStyle-VerticalPadding="0"
      StaticHoverStyle-BackColor="#6B0808"
    StaticHoverStyle-ForeColor="#FFFFFF"
      DynamicMenuItemStyle-BackColor="#f0f0f0"
      DynamicMenuItemStyle-ForeColor="#0000A0"
    DynamicMenuItemStyle-BorderStyle="Solid"
      DynamicMenuItemStyle-BorderColor="#c0c0c0"
      DynamicMenuItemStyle-HorizontalPadding="5"
      DynamicMenuItemStyle-BorderWidth="1"
      DynamicHoverStyle-BackColor="#6B0808"
    DynamicHoverStyle-ForeColor="#FFFFFF"
     width="100%"
    Height="35" >
    </asp:menu>
  </div>
 </div>
 <div>
  <asp:SiteMapPath ID="SiteMapPath1" runat="server"
   NodeStyle-Font-Bold="false"
   NodeStyle-ForeColor="#0000A0"
   Font-Size="8pt"
   CurrentNodeStyle-Font-Bold="true"
   CurrentNodeStyle-ForeColor="#6B0808"
   CurrentNodeStyle-Font-Underline="false"
   PathSeparator="&nbsp;&raquo;&nbsp;" >
   <RootNodeTemplate>
```

```
    &nbsp;&nbsp;<a href="CH21TestBreadcrumbsVB.aspx">Home</a>
   </RootNodeTemplate>
  </asp:SiteMapPath>
 </div>

 <div>
  <asp:ContentPlaceHolder ID="PageBody" Runat="server" >
   <div align="center">
    <br />
    <br />
     <h4>Default Content Displayed When No Content Is Provided
     In Content Pages</h4>
   </div>
  </asp:ContentPlaceHolder>
 </div>
  </form>
</body>
</html>
```

PREV

About the Authors

Micahel A. Kittel has nearly 30 years experience in the software industry. He has been working with Microsoft technologies for more than 10 years and with ASP.NET since the alpha release of 1.0. He has been the system architect and led the development of applications for Lexis-Nexis, Plow & Hearth, ReturnBuy, and many others. Michael has a Microsoft Certified Solutions Developer certification and is currently a managing consultant at Dominion Digital, Inc. (www.dominiondigital.com), a firm that specializes in helping companies envision and achieve maximum business value from investments.

Geoffrey T. LeBlond is the coauthor of *Using 1-2-3*, the first computer book that sold over 1 million copies. Geoff is the author of numerous computer books and was the developer of Oriel, an early scripting language for Microsoft Windows. More recently, Geoff has been focusing his attention on developing web applications using ASP and ASP.NET.

PREV

PREV

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of *ASP.NET 2.0 Cookbook*, Second Edition, is a thorny woodcock (*Murex pecten*). This carnivorous marine snail is indigenous to the IndoPacific region of the world and is commonly found in the shallow waters off the coast of Japan's sandy beaches. Averaging 13 centimeters in length, the woodcock's elongated shell contains a stunning spine of thorns, and at firs glance might be mistaken for the skeleton of a fish. While scientists are uncertain of the evolutionary advantages of this shell structure, some theorize that it serves to help ward off fish and other predators. They also believe the woodcock's needles may prevent the creature from being lodged in the soft sand and mud of its habitat.

The thorny woodcock has been immortalized in Western folklore, in which it is commonly referred to as the Venus comb or mermaid's comb. The animal's shell of needles is mythically purported to be the definitive fine-toothed comb, ideally suited for brushing even the delicate hair of a goddess.

Shell collectors are also drawn to the unique beauty of the thorny woodcock. Although its shell is not particularly rare, it is quite fragile, and a woodcock with a fully intact skeleton of thorns is a highly prized specimen for the distinguished conchologist.

Matt Hutchinson was the production editor for *ASP.NET 2.0 Cookbook*, Second Edition. GEX, Inc. provided production services. Abby Fox, Genevieve d'Entremont, and Colleen Gorman provided quality control.

Emma Colby designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from *Animate Creation*. Karen Montgomery produced the cover layout with Adobe InDesign CS using Adobe's ITC Garamond font.

David Futato designed the interior layout. This book was converted by Keith Fahlgren to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myria Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano, Jessamyn Read, and Lesley Borash using Macromedia FreeHand MX and Adobe Photoshop CS. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Sanders Kleinfeld.

PREV

Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

PREV

PREV

Index

PREV

Index

PREV

Index

Index

PREV

Index

PREV

Index

PREV

Index

PREV

Index

PREV

Index

PREV

Index

PREV

Index

PREV

PREV

Index

PREV

Index

PREV

Index

PREV

Index

Index

PREV

Index

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X]

PREV

Index

PREV

Index

PREV

Index

PREV

PREV

Index

PREV

PREV

Index

PREV

Index

PREV

Index

PREV

PREV

Index

PREV