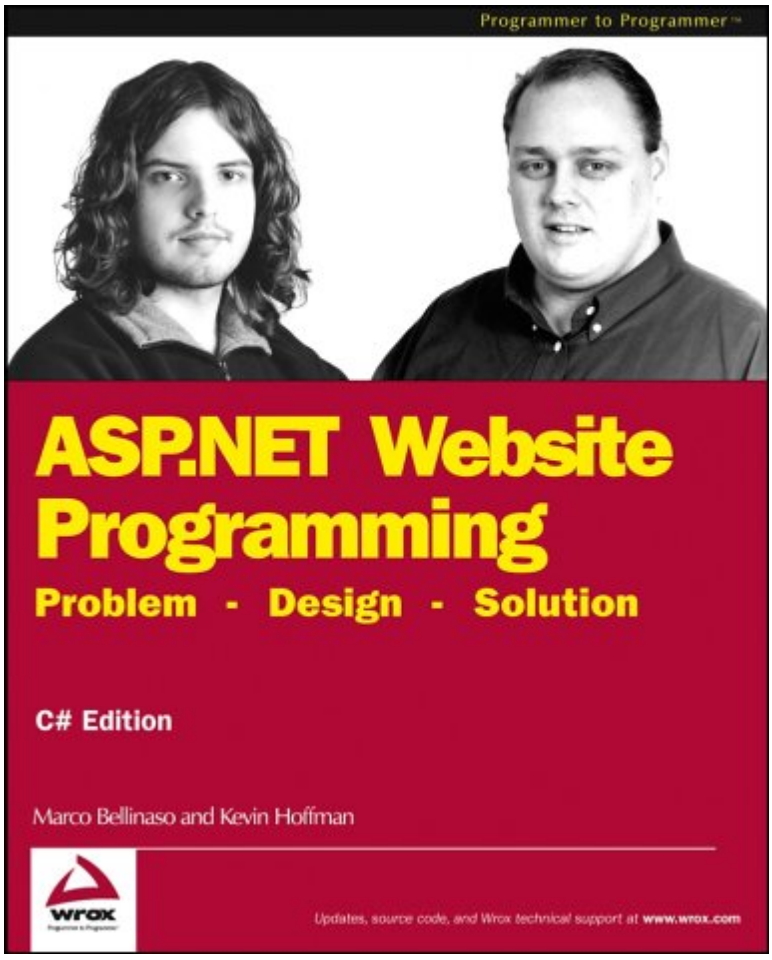
ASP.NET Website Programming, C# Edition: Problem, Design, Solution

Marco Bellinaso

Kevin Hoffman



Wrox Press Ltd.

First Printed in March 2002

Latest Reprint : November 2002

Published by Wrox Press Ltd,
Arden House, 1102 Warwick Road, Acocks Green,
Birmingham, B27 6BH, UK

Printed in the United States
ISBN 0764543776

Trademark Acknowledgements

Credits

Authors
Marco Bellinaso
Kevin Hoffman

Commissioning Editor
Dan Kent

Technical Editors
Dianne Arrow
David Barnes

Index
Andrew Criddle

Managing Editor
Viv Emery

Project Manager
Helen Cuthill

Production Coordinator
Abbie Forletta

Cover
Chris Morris

Technical Reviewers
Don Lee
Dan Maharry
Christophe Nasarre
Matthew Rabinowitz
Marc H Simkin

Proof Reader
Dev Lunsford

About the Authors

Marco Bellinaso

Marco Bellinaso is a freelance software developer. He lives in a small town close to Venice, Italy. He has been working with VB, C/C++, ASP and other Microsoft tools for several years, specializing in User Interface, API, ActiveX/COM design and programming. He is now spending all his time on the .NET Framework, using C# and VB.NET.

He is particularly interested in e-commerce design and implementation solutions with SQL Server, ASP.NET, and web services. He is a team member at www.vb2themax.com, for which he writes articles and commercial software, such as add-ins for MS Visual Studio and other utilities for VB and .NET developers.

Marco recently co-authored "Beginning C#" from Wrox Press, and is also a contributing editor for two leading Italian programming magazines: Computer Programming and Visual Basic Journal (Italian licensee for Visual Studio Magazine). Reach him at mbellinaso@vb2themax.com.

Acknowledgments

Writing this book has been a real pleasure to me. It gave me the opportunity to work with ASP.NET on a good project, and to improve my knowledge of the technology along the way. So it surely has been worth the effort! And of course, everyone likes to be published writing about what they like to do and how to do it. :-)

I owe many thanks to Wrox Press for giving me the opportunity to write the book: this is the most English I've ever written, so I guess the editors and reviewers had some extra work with me, although they were so kind as to never confess it. Some of these people are Daniel Kent, David Barnes, and Dianne Arrow.

Other people contributed to this project, in a way or another, now or in the past, and I'd like to mention at least a few names. First of all a really big thank you goes to Francesco Balena, famous speaker and author, and editor in chief of the Italian licensee of VBPJ (now Visual Studio Magazine). He reviewed and published an article about VB subclassing that I wrote some years ago, when I had no editorial experience at all. Since that moment he has continued to help me by advising how to improve my writing style, pushing me to start writing in English, suggesting the hottest technology to study, and giving the opportunity to work on some cool software projects as part of the VB-2-The-Max team. Francesco, all this is greatly appreciated!

Two other developers I work with for the Italian magazines, who helped me in different ways, are Dino Esposito and Alberto Falossi.

Giovanni - Gianni - Artico is the person who initiated me in the programming art, suggesting to start with VB and then to learn C/C++ as well. Thank you for answering my questions when I was at the beginning, and for still helping me in some situations.

A mention goes also to my closest friends. They still remember me after several "sorry, I can't come today" rebuttals, and have put up with me when I was under pressure and not the nicest person possible.

Last but not least I have to say thank you to my family, who bought my first computer and a lot of programming books when I was in high school and couldn't buy all that stuff by myself. They didn't offer much moral support during the work - mostly because they didn't have a clue of what I was doing! I kept it a secret to almost everybody - I hope it will be a nice surprise. :-)

Kevin Hoffman

Kevin has always loved computers and computer programming. He first got hooked when he received a Commodore VIC-20 from his grandfather, who had repaired it after finding it in the trash. He then started a prolific but unprofitable career writing shareware games and utilities for electronic bulletin board systems.

He started working as a programmer while still in college, writing computer interfaces to solar measurement devices and various other scientific instruments. Moving to Oregon, he did everything from technical support to tuning Unix kernels, and eventually working as an ASP programmer for 800.COM, a popular on-line electronics retailer. From there he moved on to working on large, enterprise ASP applications.

Then he finally found .NET, which he now spends 100% of his programming and learning efforts on. A big C# fan, who would use it to do everything including brush my teeth if only he could figure out how, Kevin has been writing on .NET for Wrox since the middle of Beta 1. He plans to continue until we get tired of him. He's currently in Houston, Texas sweating a lot and working on web services and other large-scale .NET applications.

Acknowledgments

I'd like to dedicate this book to the rest of my "family", without whom I could not have accomplished many of the things I am proud of today. I would like to thank Gerald for all his support - a best friend in every sense of the word - and his daughter Keely for making me laugh. I would also like to thank Jen, Jocelyn, and Emily for their support and being there for me. And as always I want to dedicate my work to my wife, Connie - without her support I would never have published a single word.

Next

ASP.NET Website Programming, C# Edition: Problem, Design, Solution

byMarco BellinasoandKevin Hoffman

ISBN:0764543776

Wrox Press 2002 (538 pages)

This book shows you how to build an interactive website from design to deployment. Packed with solutions to website programming problems, it will have you building well-engineered, extendable ASP.NET websites quickly and easily.

Table of Contents Back Cover

Table of Contents

Introduction

byMarco BellinasoandKevin Hoffman

Wrox Press ?2002

Prev Next

Introduction

Welcome to ASP.NET Website Programming. In this book we will build an interactive, content-based website using expandable, interchangeable modules. By the end of the book you will have developed your ASP.NET skills for producing effective, well-engineered, extendable websites.

ASP.NET is a great tool for building websites. It contains many built-in features that would take thousands of lines of code in classic ASP. And it does not require admin rights in order to deploy compiled components - your whole site can be deployed in one folder.

This book will guide you through the bewildering features available to ASP.NET developers, highlighting the most useful and exciting.

The book concentrates on websites that focus on content. It does not show how to produce an e-commerce system, although a lot of the advice will apply to e-commerce sites. We could add a shopping basket module using the same foundations, for example.

This book is different to most Wrox books, because we build a single working website throughout the book. However, each chapter stands alone and shows how to develop individual modules, which you can adapt for your own websites. We also suggest a framework that allows us to create modules and slot them in to the website quickly and easily.

What Does This Book Cover?

The chapters in this book follow a problem-design-solution pattern. First we identify what we need to achieve, then we sketch out how we will achieve it, and finally we will build the software in Visual Studio .NET.

Most chapters involve building a 3-tier system, with data, business, and presentation layers. We will also see how to build separate modules so that they integrate well into the whole site.

looks at the website as a whole. We identify the problem that it is trying to solve, and discuss how we will go about solving it. We then come up with a solution - which involves building and integrating the modules detailed in the other chapters.

builds the foundations of our site. We set coding standards and design our folder and namespace structure. We create our initial database - although at this stage we have no data to put in it. We also build site-wide error handling code and base classes for our data and business layer objects.

extends our foundations to the presentation layer. We will build base classes for the ASP.NET pages in the site, a custom error page, and site wide navigation, header, and footer controls.

presents a file management module, which we can use to download and upload source code for the site, and make changes online. We will also look at Microsoft's Data Manager, which enables us to manage SQL Server databases through our website.

covers user accounts. We look at how to create a powerful role-based security system, and integrate it with ASP.NET's built-in authentication features.

shows how to provide regularly changing news content through a website. We also build a web service to expose news headlines to other sites and applications, and a Windows news ticker that uses this web service.

looks at advertising. We create our advertising system by extending the ASP.NET AdRotator control to provide the power we need. We look at logging *hits* and *impressions,* and providing reports to advertisers.

covers opinion polls and voting. We look at how to administer questions, log votes, and collate them into useful reports.

provides the tools to create e-mail newsletters. We will look at how to create messages in plain text and HTML, and how to administer lists and set up new ones.

looks at forums. We create everything you need to post and read messages, and give administrators special permissions. Along the way, there is some powerful use of the DataList and DataGrid controls. We also look at how to use regular expressions to provide limited HTML support, without opening our forum to the risk of cross-site scripting.

shows how to deploy the site. We will look at the ways Visual Studio .NET allows us to provide source-free distributable versions of our software, and how to deploy our sites onto hosting services.

looks to the future. We've only just begun our lives as ASP.NET website developers and here we will look at ways in which Wrox can support your continued development. In particular this includes the book's P2P list, where you can work together with fellow readers and benefit from each other's ideas and experience.

Prev Next

byMarco BellinasoandKevin Hoffman

Wrox Press ?2002

Who Is This Book For?

The book is for developers who have a reasonable knowledge of ASP.NET, and want to apply that knowledge to building websites. You will get the most from this book if you have read a decent amount of Wrox's Beginning ASP.NET using C#, or Professional ASP.NET and a C# book.

You should be comfortable using Visual Studio .NET to create ASP.NET projects, and that you know C#.

What You Need To Use This Book

To run the samples in this book you need to have the following:

- Windows 2000 or Windows XP.
- Visual Studio .NET 1.0. We have tested the code for version 1.0, although most of the code should work in late pre-release versions. Nearly everything will also work in Visual C# .NET Standard.
- SQL Server 2000 - although most of the techniques we use could apply to any database system, including Access.

To get the site working you may also need an ASP.NET web host. We will give some guidance on choosing one towards the end of the book.

Conventions

We've used a number of different styles of text and layout in this book to help differentiate between the different kinds of information. Here are examples of the styles we used and an explanation of what they mean.

Code has several fonts. If it's a word that we're talking about in the text - for example, when discussing a For...Next loop, it's in this font. If it's a block of code that can be typed as a program and run, then it's also in a gray box:

```
<?xml version 1.0?>
```

Sometimes we'll see code in a mixture of styles, like this:

```
<?xml version 1.0?>
```

Customer Support

We want to hear from you! We want to know what you think about this book: what you liked, what you didn't like, and what you think we can do better next time. Please send us your comments, either by returning the reply card in the back of the book, or by e-mailing <feedback@wrox.com>. Please mention the book title in your message.

We do listen to these comments, and we do take them into account on future books.

How to Download the Code for the Website

It is well worth getting the website working on your own machine before reading too much of this book. It will help you follow the descriptions, because you will be able to see how code snippets relate to the whole application, and experience the modular approach first hand.

To get the code, visit www.wrox.com and navigate to ASP.NET Website Programming. Click on Download in the Code column, or on Download Code on the book's detail page.

The files are in ZIP format. Windows XP recognizes these automatically, but Windows 2000 requires a de-compression program such as WinZip or PKUnzip. The archive contains the whole site, plus a readme describing how to get it up and running.

Errata

We've made every effort to make sure that there are no errors in the text or in the code. If you do find an error, such as a spelling mistake, faulty piece of code, or any inaccuracy, we would appreciate feedback. By sending in errata you may save another reader hours of frustration, and help us provide even higher quality information.

E-mail your comments to <support@wrox.com>. Your information will be checked and if correct, posted to the errata page for that title, and used in subsequent editions of the book.

To find errata for this title, go to www.wrox.com and locate ASP.NET Website Programming. Click on the Book Errata link, which is below the cover graphic on the book's detail page.

E-mail Support

If you wish to directly query a problem in the book with an expert who knows the book in detail then e-mail <support@wrox.com>, with the title of the book and the last four numbers of the ISBN in the subject field of the e-mail. Please include the following things in your e-mail:

- The **title of the book, last four digits of the ISBN,** and **page number** of the problem in the Subject field.
-

Chapter 1 - Building an ASP.NET Website

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Prev Next

Chapter 1: Building an ASP.NET Website

Overview

In this book we are going to build a content-based ASP.NET website. This website will consist of a number of modules, which will all fit together to produce the finished product.

We will build each module in a standard order:

- Identify the **problem** - What do we want to do? What restrictions or other factors do we need to take into account?
- Produce a **design** - Decide what features we need to solve the problem. Get a broad idea of how the solution will work.
- Build the **solution** - Produce the code, and any other material, that will realize the design.

This book focuses on programming. When we talk about design, we generally mean designing the software - we will not be looking at graphic or user interface design.

Your website will not be solving all of the same problems as ours, but many of the modules we build - and the programming techniques we use - are very transferable.

In this chapter we will take a high-level look at the whole site - what it needs to do, and how it will do it.

Prev Next

byMarco BellinasoandKevin
Hoffman?

Wrox Press ?2002

Prev Next

The Problem

We will be building a website for DVD and book enthusiasts. In outlining the site's problem, we need to consider the purpose and audience. In real life this stage would be business-oriented - taking into account things like advertising demographics, competition, and availability of funding. These processes need to be analyzed rigorously, but we will leave all that to the managers.

Our site will cater for lovers of books and DVDs. It will provide useful content and try to build community. Our visitors will want to read about these things, and contribute their opinions, but each visit will be fairly short - this will not be a huge database in the style of the Internet Movie Database (www.imdb.com). It will be funded by advertising, and will rely on repeated (but fairly short) visits from its readers.

We also need to consider constraints. These are more practical. One of the major constraints that this site faced was the development team - the members would never meet, because they were on opposite sides of the world. This meant that the design must allow one developer to work on sections of the site without interfering with other developers working on different sections. But all of the sections needed to eventually work together smoothly. In most cases the separation between developers will be less extreme, but giving each developer the ability to work independently is very useful. We need to design and build methods to enable this.

Site development never really finishes - sites tend to be tweaked frequently. Another key to successful websites is to design them in a way that makes modification easy. We will need to find ways to do this.

We will call our site ThePhile.com, because it is a site for lovers of books (bibliophiles) and DVDs (DVD-philes). It's also a play on the word 'file', because our website will be a definitive source of information.

Prev Next

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Prev Next

The Design

We have outlined what our site needs - now let's look at how we can provide it. The main points raised in the problem section were:

- Enable developers to work from many different locations
- Build a maintainable, extendable site
- Build community
- Provide interesting content
- Provide revenue through advertising
- Encourage frequent visits

Let's discuss each of these in turn.

Working From Different Locations

Our developers need to work on sections of the site with relatively little communication. Our developers are in different countries so face-to-face meetings are impossible. Telephone conversations can be expensive, and different time zones cause problems.

We need to design the system so that developers can work on their own section of the site, knowing that they will not damage the work of others.

A good way to solve this is to develop the site as a series of **modules,** with each module being fairly independent. Of course there will be shared components, but changes to these will be rare and can be done in a controlled way. In this book, we work in modules. We also make frequent use of **controls.** This means that components for a page can be developed independently, and easily 'dropped in' as needed - changes to the actual pages of the site are kept to a minimum.

A Maintainable, Extendable Site

Most websites have new features added quite frequently. This means that from the start the site needs to be designed

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

Prev Next

The Solution

We've seen what we want the site to do, and sketched out some rough ideas of how we might provide it. Now we'll look at how to build our solution. This really encompasses the whole of the book. Here we'll look at how each chapter relates to our initial problem and design.

Working From Different Locations

In the next two chapters, we will provide a framework for development. This will lay down coding standards, and a framework for organizing the modules into folders and Visual Studio .NET projects.

We will decide what namespaces we will use for each module, and all the other things that will make team working as hassle-free as possible. We will also develop some initial UI features to use across the site, promoting a unified feel. These include a header, footer, and navigation control, and stylesheets.

Building A Maintainable, Extendable Site

3 will also set us on the road to a maintainable site. We will develop base classes, giving each new module a solid foundation to build on.

We will develop a web-based file manager in Chapter 4. Through this we can download and upload files, create new ones, move them, change their attributes, and even edit files online with a built-in, web-based text editor. If you've ever wanted to provide file upload facilities, offer source code for download, or provide online editing tools then this is the place to look!

Most of the modules we develop will have administration features. For these to be useful, we need to identify administrators. In Chapter 5 we will develop a user accounts system. Using this, we can collect user information and give different users different privileges. Our final site will support full role-based security, with login details stored in a SQL Server database.

Providing Interesting Content

In Chapter 6 we create a news management system. This will enable our administrators to add and edit news articles, receive and approve suggested articles from readers, and place new articles in categories. And, of course, it lets users read the news. We will create a control so that we can easily display headlines on any page that we like.

The news system will be flexible enough to also cover reviews, which will eventually form the core of our site.

Managing Adverts

Advertising will be covered in Chapter 7. We will develop a system to display adverts, and log impressions (when an ad is displayed) and hits (when an ad is clicked). This will allow us to create reports from this data to give to advertisers.

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Summary

We're now ready to look at the site in detail. Before reading the following chapters, it's worth getting hold of the code download and seeing how the final site fits together. This book does not describe every detail of the website, and it will be a lot clearer if you look at the final site first.

The code and database is available from www.wrox.com. Once you've downloaded and unzipped it, look at the readme file to see how to get it working in Visual Studio .NET. You will get far more from the book if you look at the project *before* reading on.

In the next chapter we will start to build the foundations for the rest of the site.

Chapter 2 - Foundations

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Prev Next

Chapter 2: Foundations

Overview

Laying foundations is one of the first steps we need to take when starting any non-trivial programming project. Foundations include things like code and documentation conventions, and the structure and design of the backend. In a formal development process, the foundations also typically include a vision statement of some kind, and a project plan.

Developers often have opposing views on how much work to do at this stage. Many want to sit in front of a keyboard and start coding straight away, while others want to spend weeks developing pages of rules and standards. Somewhere between the two extremes lies a fairly good medium. We don't want to get caught in an endless loop of designing, but we also don't want to write any code before we've figured out what our architecture and design is going to be like.

If we are building a house, and we build the foundations on sand, the house is likely to come tumbling down before the building is finished. On the other hand, if the ground is too hard then laying the foundations can be a major task in itself, placing unnecessary restrictions on the rest of the project.

This chapter will demonstrate a sensible compromise between the two extremes - building a solid but unrestrictive foundation for an ASP.NET website. First we will discuss the common problems facing an ASP.NET website architect in building the foundation. Then we will delve into designing a solution to these problems. Finally we'll implement these designs, and even get to work on some code. This chapter is geared towards both architects and developers alike. We will cover broad, high-level issues such as design and architecture, and we will also take a look at the code used to implement a solid foundation for an ASP.NET website.

Prev Next

Chapter 2 - Foundations

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

The Problem

Building a solid foundation can be a daunting task. It requires a good understanding of how the application will operate before we go into the detailed design of each component. If we build a good foundation, everything else will seem to fall into place. But if the foundation is poor, the site will take an extraordinary amount of work and time to complete, if it's completed at all.

Building the foundation of a website is really a collection of smaller, inter-related tasks. There are many aspects of the website's development that need to be part of the initial foundation's design. One such aspect is the development environment - for example team size and working style, and the tools that will be used to build the site. The type of team that will work on the project is an important factor in developing the foundation, as the latter should be developed to support the needs of the team. For example, a small team in a single office might work well with a fairly loose foundation, because they can easily make small changes here and there. But a large, distributed team will benefit if the foundation is set in stone, since negotiating a change could be a mammoth task. For the website in this book, the development team consisted of only two people. However, these two people were on opposite sides of the world. For this reason, the foundation needed to provide a stable basis for plugging in the different modules that each developer was working on.

In addition to the *development* needs, we need to determine the requirements of the website in its *deployment* environment. A website can have many different types of requirements, including:

- Physical - the software and hardware environment in which the final website will run. Requirements such as these typically dictate whether the website needs to be in a certain directory, or on a certain machine, or in a certain network infrastructure. Physical requirements also dictate the specific type of database to be used to support the system. We need to plan ahead for what type of system we're going to use to store our back-end data. Will it be a relational database management system (RDBMS) like Oracle or SQL Server, or are we pulling information from a mainframe, from a web service, or even from a collection of XML files? While you can code your data services tier to be as source-agnostic as possible, it isn't an excuse to spend less time on the definition of your data requirements.
- Architectural - we need to know how we plan on structuring our application. We need to know where the physical and logical separations are, and we need to consider things like firewalls and networking considerations. The website may need to be designed to support a certain type of development style, or modification by authorized third parties.
- Logical - these requirements are those that, for example, dictate that a website will consist of three layers of servicing components on top of a layer of database stored procedures.

The deployment environment includes both the server and the client browser. Many websites recommend, or even require, a particular browser in order to function correctly. Sometimes this is appropriate, but often it isn't. When laying the foundations of the site, a strategic decision needs to be made about what type and version of browser your

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

The Design

Now that we have formally defined the problem of building our application's foundation, we can begin the design process. Our design should reach a happy medium, providing enough foundation and structure to produce cohesive results, without getting so bogged down in design that we end up producing nothing.

Our discussion of the design process is going to look at some of the most common tasks in building the foundation of a website. Then we'll apply that general concept to our specific application by actually designing the various pieces of The Phile's foundation. The following list of items illustrates some of the concepts at the core of good foundation design:

- Naming and coding conventions
- Programming language choice
- Folder structure
- Designing the database(s)
- Building a data services tier
- Building a business services tier
- Providing for effective error handling
- Deployment and maintenance
- User interface design

Naming and Coding Conventions

Coding conventions can be unpopular, particularly where they are imposed on a team by a non-programmer and contain dated or restrictive rules. Every programmer has their own opinion about the usefulness of naming guidelines, coding conventions, and other code-related rules. The opinions range from those who think that any coding convention ruins the programmer's creative style, to those who thrive on the structure that conventions and standards

The Solution

Now that the foundation has been designed, we can start writing some code. To recap, a few of the things that we covered in our design were:

- Naming and coding guidelines - we set out the conventions to be used throughout the project
- Programming language choice - we chose C# as our development language
- Folder structure - we designed a namespace hierarchy and a corresponding folder structure for all the modules of the website
- Designing the database - we chose SQL Server as our core database
- Building the data services tier and the business services tier - we talked about the importance of n-tier design and architecture, and about the usefulness of creating a base class for each tier

This next section will cover the code and implementation of each tier's base class, as well as a custom exception class that we're going to build.

Of course, the implementation we create here might have unforeseen limitations that we will discover later. We might find ourselves changing these classes throughout the development. At this point, we have to continue with what we already know, and build what's best at this point.

To create the solution for this chapter, we're going to create a new C# Class Library project in Visual Studio .NET, and name it Core. We're going to make some minor changes to its properties and to the AssemblyInfo.cs file. Right-click the project and choose Properties. Then make sure that the Assembly Name property is set to Wrox.WebModules.Core, and the Default Namespace property is set to Wrox.WebModules.

The next thing we need to do is make sure that our project is **strong-named.** Since this is the first project in the entire web application, we get to make our **SNK file.** The SNK file is a file that contains a digital signature encrypted using the RSA algorithm. In order to make sure that all of our assemblies appear as though they're coming from the same vendor, they all need to have the same **public key.** The only way to accomplish this is to compile each of the assemblies against the same SNK file.

To create our digital signature file, we go to the command prompt and type:

SN-k ThePhile.SNK

Summary

This chapter has introduced the problem of coming up with the core of the website. After creating an initial design for the website, we went on to create a design for the foundation of our website. This included designing a namespace layout, a preliminary directory tree, and even specifying some coding standards and naming conventions. Then we discussed some of the core concepts of building a data services tier and a business logic tier, and the benefits of splitting functionality into three or more tiers. Finally, we discussed the design concept behind robust error handling and why it is so important to the success of a production website.

After designing the solution to our problem, we went ahead and got into the code, producing the assembly Wrox.WebModules.Core.DLL, which can be used by all facets of our website as the initial foundation from which much of the rest of our classes will be built. We even included an alternative DbObject, the ServicedDbObject, in case we want to make some data services components hosted by COM+ services.

Hopefully you've gained some of the following knowledge after reading this chapter:

- The benefits of a strong and cohesive namespace hierarchy
- The benefits of separating business logic from pure data services
- The benefits and details of robust error handling in a web application

You should also now know how to implement systems that have these benefits.

The classes we developed for the core of our solution can be compiled into the Core DLL at this point. However, as all we've done so far is build the core, we won't actually be putting this code to use until the next chapter, where we will be making use of the foundation code to help build our user interface elements.

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

Prev Next

Chapter 3: Foundations for Style and Navigation

Overview

Now that we have spent some time discussing many of the issues involved in creating a web application, and have begun building our core foundation, we can move on to creating the foundation of our front end, or **user interface (UI)**. In this chapter we will first identify the initial problem we need to solve relating to our front end. Then we will move on to designing a solution to this problem. Finally, we'll cover the actual code and implementation of this solution.

This chapter will give you a good look at some of the tasks that are typically considered part of the foundation-building, or setup, phase of website development. These include:

- Identifying and creating reusable interface components
- The purpose and implementation of a 'page inheritance hierarchy'
- The purpose, benefits, and implementation of cascading stylesheets
- Using XML and XSLT to create content that is quick and easy to maintain

Prev Next

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

The Problem

At some point we will need a front end (user interface) for our website. It would be fairly easy (especially for those programmers who've already spent a lot of time building classic ASP pages) to just open up a favorite editor and start cranking out page after page of content. I'm sure many of us have been in this situation before, which is why many of us remember the pain and suffering involved when we were told to change the layout, the style, or some other fundamental UI feature after we'd already built dozens of ASP pages from scratch. There's nothing worse than having to go back and rewrite ASP pages because of a color change or something else that should be equally trivial.

To avoid this kind of maintenance nightmare we want the UI to be simple to maintain and modify. In order to achieve this we should build the UI on a *solid foundation*. Without a solid foundation for the user interface, changes are incredibly difficult and painstaking to make, and maintenance of the front end can be a laborious task.

We also want it to be a good UI in terms of user experience. Following good usability and user interface design principles is absolutely essential to the success of your website. If the users are annoyed with the display on their screen when they see your site, they won't come back. Likewise, if they find it too difficult to get what they want from your site because it doesn't flow properly, isn't intuitive, or doesn't have clearly labeled functionality, they will also avoid your site like the plague. One thing to always keep in mind is that, no matter how good your site is, you will always have competition on the Internet.

There are many books on the market today that cover topics such as designing your website to meet the needs of your users, including User-Centered Web Design by John Cato (ISBN 0-201398-60-5). Something else you might want to take into consideration are users with accessibility needs who might have difficulty navigating a website that doesn't make certain interfaces explicitly available to them.

The Problem Statement and Requirements

Our problem has two different facets. The first is, of course, to provide a solid, functional foundation on which to build the rest of our user interface. This is actually the problem statement. The other facet, which we cannot ignore, is the requirement that our design for our UI fundamentals should strive toward the following common goals:

- Achieve maximum ease-of-use through well-planned UI elements and efficient use of web page 'real estate'. Real estate is the available space on a web page in which you can display meaningful information to a user. Examples of poor use of real estate are pages in which important information occurs in such a position as to force the user to scroll down (or off to the side) in order to see it.
- Provide maximum flexibility by allowing configuration changes of UI elements to take place with minimal or zero recompilation of code.
- Keeping in mind that a site's look and feel is almost as important as its functionality, we want to make the website attractive and intuitive to our users.

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

The Design

Now that we've determined that our problem is the lack of a solid UI foundation, we can go about designing the basics of our user interface, or presentation layer. Anyone with any experience of a full software development life cycle knows that no matter how much effort you put into an initial design, it probably won't cover every scenario. This is why many managers opt for the **Unified Process,** a very common and popular iterative process that makes allowances for changes in specification and design in the middle of a development project. Other managers, especially those producing Microsoft-based solutions, prefer the **Microsoft Solutions Framework (MSF).**

You can read more about the Unified Process in the book The Unified Software Development Process by Jacobson, Booch, and Rumbaugh (ISBN 0-201571-69-2). In addition, you can find more information about the Microsoft Solutions Framework at http://www.microsoft.com/msf.

While there are management processes that allow us to make room for changes in our design, specification, and requirements throughout the life cycle of our project, there are some things we can do in terms of code and infrastructure to make the development of those changes easier as well. Some of the common things we can do to make our website code more agile are as follows:

- Use cascading stylesheets to 'classify' different types of UI elements, such as headers, footers, tables used for certain purposes, background colors, and font styles.
- Use an object inheritance hierarchy in our component model to encapsulate functionality, properties, and visual traits common to related groups of pages and UI elements.
- Use reusable controls (server and user) in order to encapsulate common or frequently displayed UI elements that may occur on many or all pages, and to provide a code-enforced uniform look and feel.

We mentioned in the Problem section the desire to create an attractive UI. If you build a website that provides amazing functionality, but the interface is dull, drab, and uninspired, then you're probably not going to be as successful as a competitor who provides fewer services with a nicer-looking website. It is a sad, unfortunate fact. The other thing to keep in mind is that the development process is iterative, and you'll probably go through many iterations of your user interface design before any customer ever sees your website, so don't grow too attached to any one particular idea.

Typically, one of the first steps people take when designing the UI of a website is to create mock-ups or samples of what they think the website might look like during a typical user session. Remember, the purpose here is not to provide any functionality, just a foundation on which to build the UI. We did the same thing when we built this website.

Being a programmer who learned HTML using Emacs on a Unix machine, I still prefer to do my HTML design in Notepad, and so the mock-ups we used to create samples of our user interface were done in simple,

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

Prev Next

Error Handling

When we were building the core foundation of our website, in Chapter 2, we built our own custom exception class, AppException. We did this in order to have an exception class that we could throw that would guarantee the writing of an event log entry. We can also extend this class later to add e-mail functionality to our custom errors without tracking down every single line of code that throws an exception.

As you've been working with ASP.NET you've probably seen the default exception screen. It isn't exactly pretty and our application loses control at that point. What this means is that if we don't trap exceptions properly and they display using ASP.NET's default mechanism, we can't guarantee that any of our code will execute. For this reason, we want to make sure that we're exerting strict control over the exception handling system. For our design, we would like to develop an error trapping system where we control the display of the error information to the users. We aren't quite sure how we're going to accomplish this at this point, but we're certain that we need to implement some form of error trapping system in the core of our presentation tier, especially if errors are going to 'bubble up' from the business tier or data services tier at some point.

Prev Next

The Solution

This chapter has presented a problem - the need for creating a solid foundation for the presentation tier of our web application. After identifying the problem, we went on to discuss the concerns involved in designing a solution to this problem. Now that we have our basic design, and we have a good idea of what we plan on doing to create our presentation-tier foundation, let's get into the code and create some user interface elements.

We are going to be working with three projects, all of them within our main VS.NET Solution:

- ThePhile - the solution's default project, containing the site's homepage, and stylesheets. As we develop modules, it will need to reference to the presentation layer of each module. For now it should reference Core, Controls, and PhilePageBase.
- Controls - containing site-wide user controls: a header, footer, and navigator.
- PhilePageBase - containing a base class for all pages that we create for the site.

Let's look first of all at our stylesheets.

Styles

As we discussed in the design section, we know that we're going to need a stylesheet that provides user interface element templates, or **classes,** for our website. As we discussed earlier, we're going to have one main stylesheet to aid in providing a consistent look and feel, and then we'll create another stylesheet for our navigator control. To recap, the list of elements that we decided would need to be classified in our main stylesheet are as follows:

Site Header, Poll Header, Poll (Generic), Book News Header, Book News (Generic), Book News Item (Left Side, Date Field), Book News Item (Right Side, Description Field), Alternating Book News Item (Left Side, Date Field), Alternating Book News Item (Right Side, Description Field), DVD News Header, DVD News (Generic), DVD News Item (Left Side), DVD News Item (Right Side), Alternating DVD News Item (Left Side), Alternating DVD News Item (Right Side), and Site Footer.

The true benefit of these classes is that any time we decide to make a change to any of the user interface elements (this includes when sales or marketing decide to make this change, too!) all we have to do is modify the entries in the CSS file, and we don't have to worry about tracking down each and every page that displays the particular UI element we're modifying.

Here is the listing for our main stylesheet source file (ThePhile.css - this file should be created in the Styles directory directly under the main application directory, in other words ThePhile\Styles\ThePhile.css). First we have the style definition for <BODY> elements:

Summary

This chapter began with an introduction to the problem: the need to provide a clear, consistent, solid, and scalable foundation for the presentation tier of our web application. We then worked through the design of this foundation, discussing stylesheets, subclassing the default page class, creating a navigation control, creating headers and footers, and error handling within ASP.NET. After having read this chapter, you should now be familiar with the following concepts:

- Identifying and creating reusable interface components by creating user and server controls.
- The purpose and implementation of a page inheritance hierarchy, using a page base class to save time and promote consistency.
- How to dynamically render HTML using data styled with cascading stylesheets.
- How to use XSLT to transform XML into HTML and other forms of data, and the difference between XSLT and CSS.

In the next chapter we'll take the core foundation that we've been building and use it as a platform on which to build our first module, a module that allows for administration and maintenance of the site's files remotely via the web.

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

Prev Next

Chapter 4: Maintaining the Site

Overview

Any real website is generally made up of a lot of pages, images, XML/XSL files, stylesheets, databases, and other types of document. It's very common to have many hundreds or even thousands of files for a single website. During development of the site these files will usually be modified several times. This will also continue after deployment, since no application is ever really finished - particularly when we can redeploy to all our users at once. As a result, an integral part of any development work is having some kind of maintenance system.

In this chapter we'll explain why it's useful to have an online site management system, and we'll design and build one that allows us to easily maintain the site's files and directories.

Our solution will provide file uploads over HTTP connections, a useful technique that is not limited to site maintenance. For example, web-based e-mail sites use this method to upload attachments, and many community sites use it to upload images for user profiles.

We will also build an online text editor, so that we can edit our ASPX files right in the web browser.

Our tools will really be for administrators or developers to use. But with a simplified, restricted front-end we could use this technique to build a maintenance system for even the most technically inept client!

We will also present an existing third party tool, Microsoft's Web Data Administrator. This could save our developers a great deal of time managing the site's SQL Server database, both before and after deployment.

Prev Next

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

The Problem

During the development of our site, we'll need to add, copy, and move files, change the source code of ASP.NET pages, edit the stylesheets, and generally fix things here and there. Since we're working on a test machine, and as we're all familiar with doing such common operations, this does not pose any problems. Managing the SQL Server database is easy as well, because even if we don't have the program installed on our development machine, through the Enterprise Manager we can do everything we could if we had the SQL Server on our local computer.

After development will come the time to upload everything and to test the website online. We'll almost certainly need to make further changes, upload additions, move files around, and perform other file management operations. The same applies to the database: we'll need to add, edit, or delete records, run and edit stored procedures, and backup the data. If we had an in-house server, we wouldn't expect to encounter any problems here, as we would just need to move everything to the production system. Maintaining the site would be as easy as it was on our development machine. Having an in-house server offers maximum control over the system, and this is important when we need to install additional software, register COM+ components, change the IIS default settings, and so on - in fact, whenever we want to configure things according to *our* needs. However, often we do not require all this power, especially for small and medium sized sites. Also, with ASP.NET, deployment and configuration has been made much easier and flexible (take for example the use of web.config to change settings that would previously have required direct access to an IIS snap-in in ASP). Lastly, in-house or dedicated servers are expensive, and not all companies can and/or want to afford them, unless their purpose is very unusual.

Therefore, if we have budget limitations or we simply don't need full control over the system, the common solution for publishing our website is to a rent a shared server from a hosting company. We decided to choose this solution for our website, because we don't really need a deep level of system customization - web.config settings are enough for this. Also, we wanted to present an example that would be useful to the majority of readers, and that means using shared hosting.

FTP Versus Online File Management

Now that we've chosen to use a third party hosting service for our site, we should also consider the additional implications that this choice has on the ease of site maintenance. Uploading files is not a problem - we need nothing more than a simple FTP client to upload, download, rename, and perform most of the other necessary operations on the files. There are lots of them on the market, many available for free. On the other hand, using FTP to update every changed file can be slow and boring, especially when you need to upload the same very large file several times for minor changes, perhaps affecting only a single line of code each time. Sometimes FTP can be slow or even inaccessible when the server is busy (remember that we chose to use a *shared* server) or because the FTP server is temporarily down. Imagine another situation: when you're traveling or visiting a client's place without your laptop, you show the project to your client and are asked to make a quick modification. Something simple that should only take a few seconds, but how do you do it if you don't have your ASP.NET source code available, and if you don't have an FTP program to upload your changes? Often company networks have firewalls or proxies that prevent full FTP access.

Some of these issues might not seem important and you may be thinking that we could just ignore them. Admittedly, these situations are not the rule, but they *do* happen, and having a reserve plan can turn out to be a good precaution.

Chapter 4 - Maintaining the Site

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

The Design

Now that we're aware of the usefulness of having maintenance tools, let's start designing our file manager module by writing down the list of features we want to implement. A typical utility of this type includes the following functionality:

- Starting from the web root, the file manager should allow the administrator to see the list of sub directories and files, and to navigate the structure by clicking a directory name to go one level down, or an arrow at the top of the list to go one level up.
- It should display information about each file-system item (file or directory) in our application. This will include a predefined icon that describes the item type, size (of all the subdirectories and files if the item is a directory), attributes, creation date, and the date of last modification.
- There should be the ability to upload and download files.
- It should offer the ability to create, rename, copy, move, and change the attributes of any directories and files.
- It should enable us to view and edit the content of text files.

This list includes most of the basic commands that we would expect from any file manager for Windows. We want to reproduce them with a web interface running on a browser, which will allow us to perform common operations without the need for any external tools.

We want this application to respect a couple of basic requirements:

- It should be easy to integrate the tool into other existing websites
- It should not be possible for an anonymous Internet user to access the file manager; it must have a reliable authentication/authorization system

Let's look at some details to better explain the design choices.

Implementation Design

Most applications are data-driven and have a set of business rules to respect. In this case, as we said in Chapter 2, the first concern for a developer would be to decide how to split the application into several layers: data, business

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

The Solution

Now that we have a clear idea about what we're going to build, we can start creating the project with VS.NET. The files for the presentation layer are part of the main ThePhile project, and sit in the Modules\FileManager folder. Unless otherwise stated, all classes for this module should be part of the Wrox.WebModules.FileManager.Web namespace.

Classes to Work with Files and Directories

In the design section of the chapter we mentioned that the .NET Framework provides quite a lot of classes to easily manipulate and retrieve information about the file system's items. The System.IO namespace contains all the classes that have to do with the IO operations for any backing store, and some classes that allow us to do advanced stuff such as monitoring the file system and listening for changes (this was pretty hard to do with the Windows API). Since we'll use some of these classes throughout the chapter, it's worth giving a brief description of the most used IO classes:

Class	**Description**
Directory	Provides static (shared) methods for enumerating directories and logical drives, creating/deleting/moving directories and files, and retrieving/editing things like the creation date or the last access date.
DirectoryInfo	Used to work with a specified directory and its subdirectories.
File	Provides static methods for working with files: this includes opening or checking the existence of a file, and appending text data to a file.
FileInfo	Used to work with a specific file.
Path	Performs operations such as extracting the root or the file name from the specified path or combining two path strings.
FileSystemWatcher	Monitors the file system and raises events to handle changes.
Stream	Base class used to read from and write a backing store, such as the file system or network.

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Summary

This chapter presented the design and implementation of a web module, called FileManager, which provides functionality to:

- List and navigate folder contents
- Create directories
- Create and edit text files
- Download files
- Upload files
- Rename files and directories
- Modify file/directory attributes
- Delete files
- Copy and move files

This tool can help you to effectively manage your site files, resources, and directory structure. For all but very major updates, we can now rely on this tool without the need for external FTP clients or other tools.

We also saw how to set up Windows security to protect the FileManager module from unauthorized access.

Later in the chapter we installed and explored Microsoft's Web Data Administrator tool, which helps in the online management of SQL Server databases. It's particularly useful when the database serving the website is located on a remote server.

Before concluding, here are a few new features that you could add to enhance the FileManager:

-

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

Chapter 5: Users and Authentication

Overview

One of the most important aspects of a content-based website, or any website for that matter, is *community building*. As we discussed earlier in this book, a website with a strong, enthusiastic community can survive and profit far longer than other websites that might have greater funding, more development staff, or even a bigger advertising or marketing budget.

The first step towards building a thriving community is to give each user an **identity.** Each website has different needs in this area, but a few things are fairly common. The first step is to provide users with an **account** - a password-protected identity that can be used to represent the user on the website. These accounts allow us to add personalization, e-commerce facilities, targeted advertising, direct news delivery, and mailing list participation.

This chapter will first identify some of the issues and problems involved with providing user accounts. Once we've defined the problems, we will produce an initial design for a solution to this problem. Finally we will write the software to implement this solution.

We will also look at how we can secure some of the facilities and pages of our website, and how we can make provisions for administrators and power users.

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

Prev Next

The Problem

ThePhile.com provides content to users, and also allows users to contribute to the site in several ways, such as forums, which will be covered in Chapter 10. We also want our site to be able to serve different content to different users. All of these features rely on our site being able to identify its users, and determine what features they are allowed to access. In order to do this, the website must **authenticate** the user in some way, to prevent another person from using a particular user's account. For all this to work, **user accounts** will need to be created and maintained, and users will need to be correctly identified by their accounts.

Many features of the site will need to be administered remotely, so we need to allow for administrative users who will be given particular privileges. For example, some users might be able to remove offensive forum postings. However we might not want these users to access every administrative feature on the site - only those features required to moderate the forum. We might want to give another user - the main webmaster, say - access to everything. So this is more complex than simply differentiating between a set of normal users and a set of super users.

Another thing that we feel very strongly about, that should be listed as part of our problem, is the concept of **user-friendliness.** The authentication and authorization system of a content website should be as unobtrusive as possible. The users should barely be aware of the fact that the website has recognized them, and the process of logging into the website should be quick and painless.

In many other situations, such as an e-commerce website or a secured intranet application, the authentication system should be very visible, and very, very strong. In our case, however, we want the authentication system to remain in the background to prevent it from slowing down the site and confusing or distracting our users.

Now that we've described what we are trying to build, we can move on to designing a system that meets these needs.

Prev Next

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

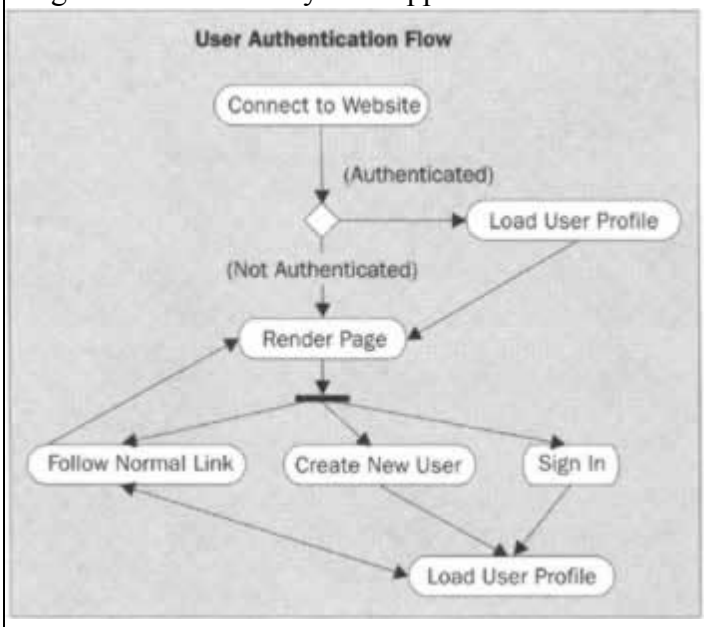
The Design

With most desktop or intranet applications we can prevent users from accessing the application until they log in. This means that the developer can assume that the user of an application can always be identified. Most websites don't work this way. Many websites don't require the user to supply a password until the very last possible minute. For example, the user is often recognized through a *cookie,* goes shopping for a while, and just after they click the Check Out button, they are prompted for their secure credentials.

Our site will follow the same policy. It will be completely acceptable for an anonymous user, or a user identified with a cookie but not authenticated with a password, to browse the site and use many of its features. If the user is identified with a cookie, some personalization can take place, as long as it doesn't compromise the user's privacy. For those parts of the site where authentication is required, the user will be forced to authenticate, otherwise they will be denied access.

In addition to building several *new* components that will drive our user authentication and security module, we should make sure that our design accounts for changes to the existing code. First of all, we're going to want to provide a link to log in, a page that allows logins, and a customized greeting in the site header that identifies the user by name.

The following UML use case diagram illustrates the process of authenticating a user. As you can see, at all times, any user (anonymous or authenticated) can follow a 'normal', or unsecured, link. This simply loads a new page and the process begins again. However, if the user chooses to authenticate, they can do this by either creating a new user account or by supplying the password information for an existing account. Finally, you can see from the diagram that it is possible to recognize a previously authenticated user without forcing them to manually authenticate. We will do this using the cookie-based system supplied with ASP.NET's forms-based authentication.



Our site will be using forms-based authentication, which is entirely based on cookies. But there are other ways to authenticate website users. For example, in order to truly secure the administrative portions of our application, we are going to use standard NT/2000 domain security (Windows authentication). This has the benefit of adding the

The Solution

We have spent some time looking at our design, and will now start to build the software. To recap, we are implementing a traditional three-tier solution with distinct presentation, business logic, and data services tiers.

We'll start by implementing the database changes, and then work our way up through the data tier and the business tier to the presentation tier.

The Database

We've already looked at the table structure that we'll use for this part of the database. We can choose whether to construct it using visual tools or SQL scripts. In Chapter 6 we'll look at the Enterprise Manager's visual tools, for now we'll present the SQL scripts that can be run in the Query Analyzer. The scripts required are as follows:

```
CREATE TABLE [dbo].[Accounts_PermissionCategories] (
                [CategoryID] [int] IDENTITY (1, 1) NOT NULL,
                [Description] [varchar] (50) NOT NULL )
 ON [PRIMARY]
GO

CREATE TABLE [dbo].[Accounts_Permissions] (
                [PermissionID] [int] NOT NULL,
                [Description] [varchar] (50) NOT NULL,
                [CategoryID] [int] NOT NULL )
 ON [PRIMARY]
GO
CREATE TABLE [dbo].[Accounts_RolePermissions] (
                [RoleID] [int] NOT NULL,
                [PermissionID] [int] NOT NULL )
 ON [PRIMARY]
GO

CREATE TABLE [dbo].[Accounts_Roles] (
                [RoleID] [int] IDENTITY (1, 1) NOT NULL,
                [Description] [varchar] (50) NOT NULL )
 ON [PRIMARY]
GO

CREATE TABLE [dbo].[Accounts_StateCodes] (
                [Description] [varchar] (60) NOT NULL,
                [StateCode] [char] (2) NOT NULL )
 ON [PRIMARY]
GO

CREATE TABLE [dbo].[Accounts_UserRoles] (
```

Summary

This chapter started off by presenting a problem that many websites today are faced with: that of identifying, authenticating, remembering, and persisting user accounts. We then went through a design for a solution to this problem. Finally, we looked at the extensive source code for an expandable, powerful implementation.

In this chapter we covered quite a few topics and took a look at a fair amount of code. We covered some important concepts, such as the uses and benefits of role-based security, and the importance of integrating into existing systems rather than creating entirely new infrastructures. We covered a complex system of assigning permissions to roles, and assigning users to those roles, and showed how we can create a fully functioning authentication system around this concept.

Hopefully, you have found this chapter useful, and you will understand the following important concepts that you can use in your own projects and sample applications:

- The issues surrounding user identification
- Issues involved in user authentication
- Issues involved in securing all or some of a web application
- Provisions for administrative or 'power' users

Now that we have our Accounts module we can move on and implement other modules that rely on it for user identification and authentication. In the next chapter we'll begin to look at modules relating to the *content* of our site, beginning with one to manage news articles.

Chapter 6 - News Management

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Chapter 6: News Management

The site we're building is basically a content site focused on DVDs and books. Content can be in the form of news, articles, reports of special events, reviews, and so on. In this chapter we're going to point out some of the content-related problems that should be considered for sites of this type. We'll then design and develop an online news manager to enable complete management of our site's content - acquiring news, adding, activating, and removing news, sharing news with others, and so on. While we will focus on managing news, many of these techniques will be relevant when dealing with other types of content.

The Problem

There are several ways to gather information and news for our site's content: we might hunt for news ourselves, get news directly from the users (a great example of this is the Add Your News link at www.aspwire.com), or rely upon a company, such as Reuters, whose business it is to gather and organize news and distribute it to third party sites. Another common technique is to keep an eye on other news sites, and scrape articles together from information available on the Internet. Many sites (for example www.wired.com) also provide links to suggested stories elsewhere. It doesn't matter which methods we use, we still need fresh and updated content if our site is to be successful and entice users to return. No user will come regularly to a site if they never, or very seldom, find some new content.

Once we have our news sources, a second problem arises: how to add news and articles to our site. We can immediately rule out manually updating or adding static HTML pages - if we have to add news several times a day, or even just every week, creating and uploading pages and editing all the links is not practical in most cases. In cases where it is practical, we don't need to write a new management system!

For our site, we need a much more flexible system, one that allows the site administrators to easily publish news without requiring special HTML tools or a knowledge of HTML. We want it to have many features, such as enabling us to organize news in categories and show abstracts, and allowing the site users to post their own news. We'll see the complete list of features we're going to implement in the Design section. For now it's sufficient to say that we must be able to manage the content online, without any other tool. Think about what this implies: you can add or edit news as soon as it is available, in a few minutes, even if you're not in your office and even if you don't have access to your own computer; all you need is a connection to the Internet and a browser. And this can work the same way for your news contributors and partners. They won't need to e-mail the news to you and then wait for you to publish it - they can submit and publish content without your intervention (although in our case we will give the administrator the option to approve or edit the content before publication). A good example of this is www.codeproject.com, which provides article categories plus an "unedited section" for the user-submitted articles that have not been edited yet.

For small content sites, or for sites where news and articles are not the main business, simply showing news and other resources is sufficient. But others may want to take this one step further, and decide to offer the news they have gathered/written and organized to other sites. Such sites might not want to spend time finding and formatting articles, may not have people who look after a news management system, but might want to fuel their site with some fresh content automatically updated on a regular basis. In many cases, paying a fee for such a service can be relatively cheap and very easy, since the site administrator doesn't have to worry at all about the content, and they'll be able to focus on the site's main business (commerce of goods related to the published news, for example). Therefore the

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

The Design

In this section we're going to work on the design of our online tool for acquiring, managing, and sharing the news content of our site. Specifically we will:

- Provide a full list of the features we want to implement
- Design the database tables for this module
- Write down a list and a description of the stored procedures that provide access to the database
- Describe the object models of the data and business layers
- Describe the user interface services specific to news management, such as the site pages, reusable user controls, and web services
- Explain how we will ensure security for the administration section and for other access-restricted pages

Features to Implement

Let's start our discussion by writing down a list of the features that the news manager module should provide in order to be flexible and powerful, but still easy to use. We might decide to add more features later, but these are the things we definitely need to implement:

- A news item can be added to the database at any time, with the option not to publish it until a specified release date. In addition, the person submitting the news must be able to specify an expiration date, after which the news will be retired. If these dates are not specified then the news should be immediately published and remain active indefinitely.
- News items or articles can have an approved status. If it is the administrator who submits the news item, it should be approved directly. If we allow other people, such as staff or users of the site, to post their own news, then it should be added to the database in a "pending" state. The site administrator will then take care of controlling this content, applying any required modifications, and finally approving the news for publishing once it is considered suitable.
-

The Solution

We've discussed just about every aspect of the design, and we are now ready to produce the solution. We'll follow the same pattern as in the *Design* section: from the creation of database tables and stored procedures to the implementation of security, passing through the data access and user interface services coding.

Working on the Database

Creating the database tables is straightforward, using the Enterprise Manager, so we won't cover it here. The best way to set up the database is to use the backed up version in the code download.

Now let's look at how to create the relationships between the tables and write some stored procedures, although these already exist in the backup.

Relationships Between the Tables

We create a new diagram in the Enterprise Manager, and by following the wizard we add the News_Categories, News_News, and Accounts_Users tables. As soon as the three tables are added to the underlying window, the Enterprise Manager should recognize a relationship between News_Categories and News_News and automatically add a connection with the correct properties. However, if it does not, click on the News_News table's CategoryID field and drag and drop the icon that appears over the News_Categories table. Once you release the button, a dialog with the relationship's properties appears. You should set the options as shown in the screenshot below:

The Cascade Update Related Fields option ensures that if we change the CategoryID primary key in the

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Prev Next

Summary

In this long chapter we have seen how to build a complex and feature-rich module to completely manage the site's news and articles. We've provided numerous features through the chapter:

- A tool for managing the database.
- Pages for browsing the published content.
- Integration with the Accounts module to secure the module and track the authors of the news items.
- A user control for showing the headlines on the homepage or in any other page.
- A web service that can share the headlines with any site or program that can use and understand XML/SOAP messages.
- A Windows client program that uses the web service to show the headlines of all the available categories, and that opens a browser and shows the whole news item when a title is clicked.

This system should be flexible enough to be successfully plugged into many real world applications. When customized to fit a particular site's needs, it could become very powerful.

Prev Next

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

Chapter 7: Advertising

Overview

No matter how wonderful the website, how amazing the look, how talented the programmers, or how useful the service - a site will not last long without a consistent source of funding. Some sites are maintained by volunteers who donate their own time and money, and others get subscription fees from readers. Many of these sites try to supplement their revenue with advertising, and other sites rely on it entirely.

Many sites use banner advertisement-style features even when they don't sell advertising space. For example, some sites exchange links with other sites. Others (for example www.play.com) use banner advertising to provide links to areas of their product catalogue that the viewer would not necessarily go looking for.

In this chapter we will present an overview of the problems involved with advertising and some common design patterns in overcoming those problems. We'll develop a reusable advertising module that we'll plug into our website and that we can easily adapt to our future needs.

The Problem

There are two main concepts you need to be familiar with to provide advertising on a website: impressions and hits.

Impressions

An **impression** occurs when a particular advertisement is displayed on a web page to a single user. Website owners, administrators, and marketing managers should consider impressions a commodity - chips to bargain with when obtaining lucrative advertising contracts.

Consider this scenario: a popular content website receives over a million impressions a day. For the sake of example, imagine that it is a fan club website dedicated to a popular video game. Its audience consists of a wide range of browsers from older adults to young teenagers and pre-teens. An impression on this site would be considered an extremely valuable commodity to a video game manufacturer. However, a manufacturer of a popular brand of craft glue would probably not find those impressions particularly valuable.

Impressions are *yours* to sell, and you should look for a buyer for your impressions that would be most interested in your target audience. Therefore, you *must* know the audience of your website in order to market your impressions to the right audience.

Typically, impressions are sold in bulk. A typical advertising contract will consist of the advertising buyer paying for a mathematical or statistical guarantee that a certain number of impressions of a given ad will be displayed in a given time period (usually measured in weeks or months).

Hits

While impressions are yours to sell, **hits** are to be considered rewards or prizes. An impression occurs and an advertisement appears on the user's page. If the user is particularly interested in what the advertisement has to offer, then the user will click the advertisement. This is called a hit. Just to confuse things more, virtually every dotcom company has a different term for hits, including clicks, click-throughs, scores, buy-ins, and more. No matter what term is used, it all boils down to the simple act of the user clicking on the advertisement.

This is where advertising gets more profitable. Hits are worth more than impressions. Advertising contracts typically pay percentages of sales resulting from banner advertisement hits, or they pay a flat rate per hit.

When designing an advertising system we need to keep track of the following:

- Impression counts
- Hit counts
-

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

Design

Now that we have specified the broad requirements of our advertising system, let's lay down a design for it. Once again ASP.NET provides a component that almost does what we want - the AdRotator. So before we look at the components we are going to build, let's look at how the AdRotator works and what impact using it will have on the rest of our design.

Using the AdRotator

The AdRotator handles a lot of the work associated with displaying an advert. It will save us time overall if we use it, but it has its own foibles that we need to overcome.

We would like to store all data for our application in the database, but AdRotator only supports an XML file. So we will need to find a way to work around this. Information required by the AdRotator will go in the XML file, while related data will go in the database. We will need a way to tell which entry in the database relates to which entry in the XML file.

Another limitation is that the AdRotator doesn't come with the ability to record hits or impressions. It doesn't expose a click event that we could use ourselves to record hits, so we'll have to implement this functionality ourselves. However, it does fire an event every time an ad is displayed - so we can use this to record an impression.

The XML file for an AdRotator stores the following details:

Property	**Description**
ImageUrl	Absolute or relative URL to an image to be displayed as the banner advertisement.
NavigateUrl	The URL of the page to go to if a user clicks the ad.
AlternateText	Text to display in the images ALT attribute.
Keyword	An optional keyword for the advertisement, to allow filtering. We won't be using this feature here, consult the MSDN documentation for more information on this property.
Impressions	This number indicates the *relative* importance of this advertisement with respect to all other ads in the rotators XML file. The larger this number, the more frequently the ad will be displayed. This number will typically be adjusted based on how much the advertiser is paying.

The Solution

As with any project, we hope that by the time we are ready to code we have created a good, solid, effective design. In our design we covered the data services and business layer classes that we would need to create. We also discussed the fact that we would be using the AdRotator control.

This is actually a very simple control to use, and is the core of our advertising solution. You just put the AdRotator control onto your ASP.NET page and indicate where it can find the advertisement files - it then handles everything else. In this section we'll show you how we use this control to record impressions and hits, two things that it doesn't have built-in support for recording. We'll also show you how we have implemented the rest of our design to support the advertising solution.

The Database Tables

Before building our database, we need to decide on the details of the tables. Here is a fuller schema, which we will use to build the database.

AdsManager_Advertisements Table

Column Name	Data Type	Description
AdvertisementID	int - Identity primary key	Unique ID for the advertisement.
Description	varchar	Description of the advertisement.
CompanyID	int foreign key	ID of company owning the advertisement.
Active	bit	Boolean flag indicating whether the advertisement is active.
TrueNavigateUrl	varchar	URL where the user will be redirected from the pass-through page.

AdsManager_Companies Table

Column Name	Data Type	Description

Summary

Advertising is an interesting and exciting facet of doing business on the web. Many websites pay their bills by offering advertising. Website owners and administrators can track the demographics of their audience and use that information to convince advertisers that paying for impressions on their site is a worthwhile investment. Those same administrators can also keep track of how many impressions and clicks a particular advertisement receives and use that information to bill the client (advertiser).

After reading this chapter, you should have a solid understanding of:

- Storing information on advertisements by aggregating both XML and RDBMS data in the data layer, hiding the distinction from the business layer.
- Keeping audit trails of clicks and impressions by trapping events in the AdRotator control and doing some data warehousing in the database.
- Providing sufficient information to advertisers to prove ad effectiveness (or lack of) by providing a stored procedure designed to pull reporting information from the data warehouse.

So far we've looked at several ways of providing our site visitors with information. In the next chapter we'll show how to find out what our readers think, by implementing an opinion polls module.

Chapter 8 - Polls

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Chapter 8: Polls

In this chapter we'll discuss polls - which comprise a question with a set of optional responses that the user can select from and vote for. First we'll recap why polls are useful and important for different websites. We'll then demonstrate how to design and implement an easily pluggable and maintainable voting module for our ThePhile.com site. We'll also show how to make it accessible to external clients, such as other sites or Windows programs, via a web service.

The Problem

We briefly discussed the benefits of polls in Chapter 1. To recap, sites that provide a poll usually do so because they are interested in what people think. They use polls as a form of user-to-site communication because they want views on the products they sell or review, or opinions about the market in general, or they want to know who the users are, their age, their occupation, and other demographic information. Good polls always contain targeted questions that can help the site's managers to know who their users are and what they want to find on their site. This information can be used to identify which parts of the site to improve.

Polls are valuable for e-commerce sites too, because they can indicate which products have higher interest and demand. Armed with this information, e-commerce businesses can highlight those products, provide more detailed descriptions or case studies, or offer discounts to convince users to buy from their site.

Another use for the information is to attract advertising revenue. If you look on any middle to large site the chances are that you'll see an "Advertise with us" link, or something similar. On that page you'll probably find information about the age of the typical users, the regions or countries they live in, and sometimes also their average income. This information is often gathered by direct or indirect polls. The more details you provide about your typical audience, the more chance you have of finding a sponsor to advertise on your site.

Another benefit is user-to-user communication. Users generally like to know what their peers think about a product or a subject of interest to them, and maybe even how much they earn! I must admit that I'm usually curious when I see a poll on a website. Even if I don't have a very clear opinion about the question being asked, I vote, often because I want to know which is the most popular response! This explains why polls are usually well accepted, and why users generally vote quite willingly.

Another reason why users might be willing to vote is that they may feel that their choice has some significance for the people behind the scenes. And the votes actually are important; as we've seen, the results can drive the future content of the site and other decisions.

We want the benefits of a poll facility for ThePhile.com, and therefore want to implement some form of poll on the website. Now we should consider some further details about web polls, namely the problems that we must address to successfully run a polling system.

First of all, as for the news and other content, the same poll shouldn't remain active for too long. If we leave the same poll on the page for, say, two months, we might gather some more votes, but we risk losing the interest of users who voted early on. Neither can we keep a poll up for just a couple of days, at least not if we want to achieve significant

Chapter 8 - Polls

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

The Design

As usual, in this section we're going to work on the design of the solution. We'll be looking at how we can provide voting functionality for our site. This module - like most of the others presented in the previous chapters - stores the data (questions, answers, votes, etc.) in the database shared by all modules of this book. To easily access the database we'll need a set of stored procedures and a data access layer, and a business layer to keep the presentation layer separate from the database and the details of its structure. Of course there will be some sort of user interface that allows the administrators to view and manage the data through their favorite browser.

To start with we'll list the features we want to implement, then we'll begin to design any database tables, stored procedures, data and business layers, user interface services, and security that we need for this module.

Features to Implement

Let's start our discussion by writing down a list of features that the polls module should provide:

- In order to easily change the current poll and add or remove questions, the administrator will need an access-protected administration console. It should allow multiple questions, and their options, to be added, edited, or removed. The ability to have multiple questions is important, because we might want to have different polls in different sections of our site. The administration pages should also show the current results for each option, and the total number of votes for each question.
- A user control that builds the poll box that can be inserted into any page. The poll box should display the question text and the available options (usually rendered as radio buttons to allow only one choice). Each question will be identified by a unique ID, which should be specified as a custom property for the user control, so that the web master can easily change the currently displayed question by setting the value for that property.
- We should prevent the user from voting multiple times for the same poll. Or, even better, we should be able to dynamically decide if we want to allow the user to vote more than once, and specify the period for which they will be prevented from voting again. We'll further discuss why we might want to do this in the next section.
- We can have only one poll (question) declared as current. When we set a question as being current, the question that was previously the current one should change its state. The current poll will be displayed unless we specify another question ID for the poll box. Any non-archived poll can be displayed. Of course we can have different polls on the site at the same time depending on the section (one for DVDs and one for books, for example), but setting the default question is useful because we'll be able to add a poll box without specifying the ID of the question to display (through the custom property mentioned above). We'll also be able to change the question through the administration console, without manually changing the page and

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

The Solution

Now that we have a thorough design for the module, we can start the hands-on part - the implementation of the solution. We'll follow the same order as we did in the design section: starting with the implementation of the database tables, we'll move on to the data, configuration, and business assemblies, and finally we'll look at the presentation layer. This comprises the administration section, the poll user control, the web service, and the Windows web service consumer.

Working on the Database

With the help of the Enterprise Manager, creating the tables for our SQL Server database is such a simple task that it is not necessary to describe it in detail here. Earlier in this chapter we presented the complete lists of the columns for each table, along with the most significant properties, so you should have no problems creating them. (Alternatively, script files to create the tables are available in the code download, as is a complete backup of the database.)

You can create the new tables (or even a new database if you need to) right within Visual Studio .NET, without opening SQL Server. On the far left of the IDE, near the Toolbox, you find another tab called Server Explorer. This window allows the developer to explore the server components such as SQL Server databases, events logs, message queues, etc., through an easy-to-browse tree control. Under the SQL Server leaf are listed all the available databases. If you select and expand one you can see its tables, stored procedures, views, diagrams, and functions. You can edit or delete existing objects, or create new ones, by clicking the respective commands of the popup menu that appears by right-clicking on a tree item. The following screenshot shows the IDE while displaying the data of the Polls_Options table:

Summary

This chapter has presented a working solution for handling multiple dynamic polls on our website. The complete Polls module is made up of several parts:

- An administration console for managing the polls through a web browser
- Integration with the Accounts module to secure the administration pages
- A user control that enables us to show different polls in any page we want with just a couple of lines of code
- A web service that can be accessed by external clients to get the current poll's question and results
- A Windows client program that uses the web service to display the current poll and its results

This module can easily be employed in many real world sites as it is now, but of course you can expand and enhance it. Here are just a few suggestions:

- Add more styles properties to the poll control, and the ability to remind the user which option they voted for. Currently they can see the results, but the control does not indicate how they voted.
- Add a ReleaseDate and ExpireDate to the polls, so that we can schedule the current poll to change automatically.
- Provide the option to allow only registered users to vote. Alternatively, keep the vote open to all, but allow only registered users to see the results and/or the archive page.
- Expand the web service and the respective Windows client so that it can display all the archived polls as well as the current one.

In the next chapter we're going to continue the development of ThePhile.com through the addition of another easily pluggable module. This new module will be used for managing multiple mailing lists and their subscribers, and for sending out newsletters.

Chapter 9: Mailing Lists

In this chapter we'll discuss the design and implementation of a complete mailing list system. This will enable users to subscribe to receive regular newsletters, and will allow administrators to manage mailing lists and newsletter content. First we'll look at what mailing lists and newsletters can offer to websites like ours, and will consider various ways of making the mailing list administrator's life as easy as possible. By the end of the chapter we'll have developed a flexible mailing list module that can be plugged into most sites. In keeping with the rest of the project, we will aim to integrate this module into our ThePhile.com site through the reuse of familiar code and architecture where appropriate.

The Problem

Throughout this book we've mentioned that the key to a successful site is having good content. This content also needs to be logically organized to ease navigability, have an attractive design, and offer some interaction with the user. The content not only has to be interesting and accurate, but to ensure that users keep visiting the site it must always be fresh and regularly updated. To help us achieve this for our ThePhile.com website, we built the NewsManager module (in Chapter 6) to allow an administrator to easily manage and publish new content (for example an article, a new product for sale, or a new design).

But even if fresh content is frequently added to the site, not every user will be aware of it. They might not visit the site daily or weekly just to see the latest updates, especially if the site is updated on a random basis with no public announcement of when new material has been added. A good way to inform the user that some new content has been added to the site is to send an e-mail **newsletter** that lists all the new resources available on the site. Many sites offer the option of subscribing to a **mailing list,** which typically represents a group of users interested in a certain kind of news. A newsletter is sent to a mailing list to inform the community of users that the site is still worth visiting.

You should always keep in mind that you're offering a service to the user, and it must be a high quality service if you don't want to lose your subscribers. This means that you need to provide *targeted* content. So, you shouldn't send out a general content newsletter if you have a large site with different sections and different types of content. For example, at ThePhile.com we cover DVDs and books, and so we have news for two different topics. We should provide at least two different mailing lists, so that the users can get what they want and avoid getting what they are not interested in, which they may perceive as spam.

It's a nice touch to personalize every e-mail message, for example with the name of the subscriber if this information is available, because this can help to build a more personal relationship with the subscriber. However, in order to persuade users to subscribe to a mailing list the subscription process should be as straightforward as possible and they should not be forced to provide additional personal details. The most common way of subscribing to a list is to type your e-mail address in a form on the home page and press submit. This will allow only the e-mail address to be used to personalize the newsletter. In order to achieve more extensive personalization you have to ask the users to provide more details, such as their first and last names. For your websites the choice is up to you, but we want our module for ThePhile.com to support both a basic and extended subscription form, so that we can cater for users with different attitudes and so that managing the mailing list is as straightforward as possible in either case.

Some website visitors don't like to submit their e-mail address even to be informed about changes to the website. To

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

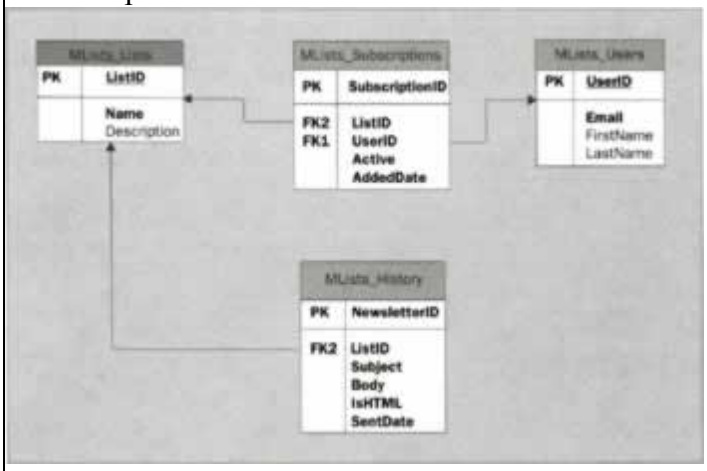
The Design

From our requirements it's clear that we're going to be handling data about users and the lists they subscribe to. In keeping with the rest of the site we'll use our SQL Server database to store this information. We'll also need an interface for administrators to manage the lists, and an interface where users can subscribe. As with other modules we'll use a multi-tier approach and will make use of data access and business logic layers to keep the UI and database separate.

In this section we'll design the database tables for this module, and design the data and business layers that we'll use for managing the subscriptions from the administration console. The nice thing about the administration console is that we won't need to manually manage the group of subscribers - the application will automatically add or remove them to/from the database. For this we will make use of stored procedures called by methods in response to certain events.

Designing the Database Tables

As discussed in Chapter 2, all modules in our project require a prefix for the tables and stored procedures. For this module it will be MLists_. We need the four tables shown in the following diagram, which also indicates their relationships:



The MLists_Lists Table

The MLists_Lists table is used to store the information about each available mailing list:

Column Name	Type	Size	Allow Null	Description
ListID	int - Identity primary key	4	No	The unique ID for the mailing list.
Name	varchar	50	No	The name of the mailing list.

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

The Solution

This module has a very similar structure to that of previous modules. The administration console has similar pages: the page for the mailing lists (questions or categories in the previous chapters), the page for the subscribers (options and news in the previous chapters), and the page for the settings. We won't list the code for these pages, as it is so similar to the code already seen - all of the code can be found in the download. The same applies to the data component. Thanks to the DbObject base class of the Core assembly, we can use the RunProcedure method to run the stored procedures and return a dataset or an output parameter. As we saw in the *Design* section, our data assembly has classes that almost exactly map the stored procedures, and there is nothing new to add that wasn't already discussed in previous chapters.

So what *are* we going to see? We'll look at the code of some of the stored procedures (to better understand the relationships between the tables), the business classes, the subscription box, and the pages required for subscribing.

Working on the Database

Looking at the diagram and the descriptions of the tables we provided earlier, you should have no problems building the tables for our SQL Server database and the relationships. (All three one-to-many relationships should have the Enforce Relationship for INSERTs and DELETEs options selected.) However, as for the previous chapters, you can go to the Wrox site and download the database backup with all the tables, stored procedures, triggers, and also some data that will allow you to test the module straight out of the download.

Creating the Stored Procedures and Triggers

In this section we won't cover the code of all the stored procedures, since in previous chapters we've already seen how the procedures for deleting and updating a record work. However, we'll show a few of the procedures for working with subscribers, because they are a bit more complex as they involve joins and relationship constraints.

Retrieving Subscriptions

Here's the code for the sp_MLists_GetSubscriptions procedure. It joins the MLists_Subscriptions and MLists_Users table, to return either all the details of the subscriptions for the specified mailing list, or just the details of those users with an active subscription:

```
CREATE PROCEDURE sp_MLists_GetSubscriptions
            @ListID int,
            @ActiveOnly bit
AS

IF @ActiveOnly = 0
  BEGIN
  SELECT SubscriptionID, MLists_Users.UserID, Active, FirstName, LastName,
                              Email, AddedDate, ListID
    FROM MLists_Users
```

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Summary

The aim of this chapter was to show how to design and build a full-featured mailing list manager that could be used as-is for many websites. We've implemented an administration console that allows us to:

- Add, edit, and delete mailing lists and their associated subscriptions.
- Create and send newsletters (both in plain text and HTML format) and consult the archive of previously sent messages.
- Modify the settings online.
- Auto-generate the HTML code for the subscription box.

Administration aside, we've developed the page that actually handles the subscription process. This used the business components developed at the beginning of the implementation section. Finally, we've used the Accounts business classes to ensure that only users with the appropriate permissions can administer the data and send out newsletters.

By doing all this we've learnt some new techniques in this chapter, most notably:

- How to use the SmtpMail and MailMessage classes to send e-mails with the Windows SMTP Server.
- The use of regular expressions to validate e-mail addresses (or, in general, any other string) against a given pattern.

As for any other module presented in this book, you can add further features to make it even more powerful. Here are just a few ideas:

- Add the ability to create and handle subscription forms that allow the user to subscribe to *multiple* mailing lists in one step, giving their e-mail address only once. This would simply require a few changes to the HTML form that you paste in your pages, and to the Subscribe. aspx page that processes the data posted to the server. Adding a subscription to more than one list is really just a matter of a few lines of code, thanks to the business layer classes that handle the actual subscription process.
- Allow each user to choose whether to receive the newsletters in HTML or plain text format, and send different newsletters according to that choice.

Chapter 10 - Forums and Online Communities

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Chapter 10: Forums and Online Communities

A successful site should build a community of loyal visitors. Internet users like to feel part of a community of people with the same interests, to discuss their favorite subjects, and to ask questions and reply to those of others. Community members will return often to meet other people that they've already chatted with, or to find comments and opinions about their interests. In this chapter we'll outline some of the advantages of building such a virtual community. We'll then identify the goals for our website community, and step through the design and implementation of a new module for setting up and managing discussion boards (forums).

The Problem

User-to-user communication is important in many types of website. For example, in a content-based site relating to programming resources, programmers need to ask questions about problems they are facing and hear suggestions from their peers. e-Commerce sites benefit from allowing users to review products online.

Two ways to provide user-to-user communication are opinion polls and discussion boards. We looked at opinion polls in Chapter 8. In this chapter we will talk about discussion boards, or **forums.**

Forums act as a source of content, and provide an opportunity for visitors to participate and contribute. Visitors can browse the various messages in the forums, post their questions and topics, reply to other people's questions, and share ideas and tips.

For our ThePhile.com site, we can offer discussion boards about books and DVDs. We will also want sub groups within that, so that it is easier for visitors to read about what they are specifically interested in. For example, if we have a books category we can have sub forums for programming books, classic literature, novels, and publishing.

Early web-forum systems often threw up long lists of messages on a single page, which took ages to load. We want to avoid this by displaying lists in pages, with each page containing a set number of messages.

Our website already has a way to identify users (the Accounts module we developed in Chapter 5), and our forums should support that. But we should also give users the opportunity to create a **profile,** so that they can post messages without revealing their true identity. These profiles should include a public name, an avatar image (a small picture that represents the user), a signature, and a homepage URL.

The administrator must be able to add, remove, or edit categories, forums, topics, and replies, and change the module's settings. If you're wondering what the difference is between a category and a forum, let's say that a category is a container for multiple forums. For example, we can have two categories: Books and DVD. The Books category can contain the following forums: programming books, books for children, school course books, spy stories, etc. The DVD category, on the other hand, will have child forums such as PC games, action movies, horror movies, cartoons, etc. In practice, a category allows better organization and keeps together forums with related subjects.

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

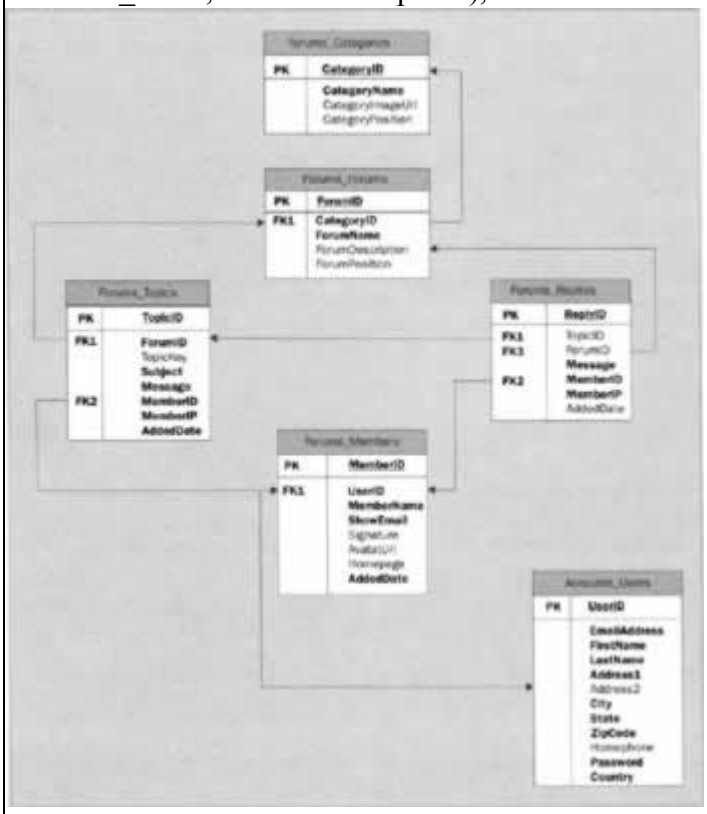
Prev Next

The Design

As usual, we will design our data layer first and work up to the presentation layer. The configuration system is quite important in this module, so we will look at the design for that before considering the business layer.

The Database

The design begins with the database tables needed to store all the categories, forums, topics, replies, and members. The prefix for the tables and stored procedure for this module is Forums_. We have five new tables (we also use Accounts_Users, created in Chapter 5), shown in the following diagram:



In the next sections we'll be explaining each table in detail, but remember that - as usual - you can download the complete database backup with the required tables, stored procedures, views, etc. from the Wrox site.

The Forums_Categories Table

The Forums_Categories table is used to store the name and image for the forum categories. It is structured as follows:

Column Name	Type	Size	Allow Null	Description
CategoryID	int - Identity primary	4	No	The unique ID for the

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Prev Next

The Solution

We've already developed several modules throughout the book, and we know the path to follow. We start by creating the database tables and the stored procedures, then we continue with the data and business classes, and finish with the ASP.NET pages for the presentation layer. In practice, we'll follow exactly the order of the design section. Much of the code required here is very similar to that of previous chapters, which was shown in great detail. So we'll skip the discussion of the common code, and try to focus on the new stuff.

Creating the Database Tables

We create the tables and relationships, and set all relationships to enforce cascade updates and deletes - except for the relationship between Forums_Replies and Forums_Members. This is because we already have a relationship between Forums_Topics and Forums_Members, and one between Forums_Replies and Forums_Topics.

When a member is deleted, all topics by that member are deleted, and all the replies of that topic are deleted as well. SQL Server can't handle a cascade delete of the same row when rows from two different tables are deleted, which would happen if we enforced a cascade update between Forums_Replies and Forums_Members. And exactly the same applies for the cascade update. So, when a member is deleted its topics are deleted, the replies of those topics are deleted, and we must ensure that the replies authored by that member for other topics are deleted as well. This can be done using a DELETE trigger, created on the Forums_Members table as follows:

```
CREATE TRIGGER DeleteForumsReplies ON [dbo].[Forums_Members]
  FOR DELETE
  AS

   DELETE Forums_Replies
     WHERE MemberID = (SELECT MemberID FROM Deleted)
```

Deleting all records of a user is an extreme measure. In most cases we would prevent a user from accessing the site by revoking all their permissions, but would leave their database entries intact.

Creating the Views

We now need to create views to add dynamically calculated columns, and create standard joins between tables.

The v_Forums_Forums View

This view joins the Forums_Categories and Forums_Forums table to return rows with information on the forums and their parent category as well. It also adds these calculated columns:

- ForumTopics: the total number of topics for the forum
-

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Summary

In this chapter we've built a forums system from scratch. We've seen how to integrate other modules such as the Core and the Accounts modules, as well as ASP.NET's built-in authentication. Our Forums module supports multiple categories and sub-forums, displays topics and replies through custom pagination, enforces the user currently registered to the site to also create a forums' profile with their username, signature, avatar, and homepage, and supports or prevents the use of HTML code and other special tags.

We've written quite a lot of code to fuel the discussion board system. Below we just list the most important techniques that this chapter should have taught you:

- Creating nested data bound controls: in the Default.aspx page we showed how to create a DataGrid inside a DataList to represent a parent-child relationship.
- Custom implementation of the Delete command for the DataList and DataGrid in Default.aspx, namely the JavaScript popup box that asks for a confirmation and calls __doPostBack to generate an event on the server.
- How to implement the DataGrid's custom pagination, by using a stored procedure that returns the specified page of records, the DataGrid's AllowCustomPaging property, and custom controls to navigate through the pages.
- Using regular expressions and the RegEx class to extract and replace string patterns.

However, we've really only scratched the surface of the features we could implement for a professional forum. Below we list some of the features they offer, in case you want to enhance our module with some more advanced functionality:

- Email notification of forum activity, or even e-mail message digests - eventually we could integrate the web forum with an e-mail discussion list.
- Add some kind of preview or revision system for users to edit their own messages.
- Banning certain words, and using regular expressions to replace them with acceptable alternatives.
- An administration console, that allows administrators to browse members, edit their profiles, or ban them

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Prev Next

Chapter 11: Deploying the Site

Overview

Now that we have developed our website, we need to **deploy** it - prepare and distribute the site so that users can access it.

The release of ASP.NET forces us to reconsider many preconceptions about the deployment of websites. For example, we need to get used to the possibility of running multiple copies of the same site on a single server, sharing different versions of identically named DLLs. Another thing that developers might find incredible is XCopy deployment, which allows a developer to deploy an application by simply copying files to the target location. There's no need to use the Registry or any complex COM registration.

In the past, deploying a large-scale web application could become a nightmare. Most enterprise web sites were comprised of dozens (or more) COM and COM+/MTS components. Maintaining the information on all of those components in the Registry and making sure that the information was updated properly when upgrading to a new version was an incredibly difficult task. ASP.NET allows entire websites to be configured with simple XML text files, and components to automatically register themselves in COM+. There's no need to look to the registry for anything in deployment of ASP.NET, completely alleviating one of the biggest ASP deployment headaches.

This chapter will discuss the general issues surrounding the deployment of ASP.NET websites and the various approaches we can take. We will describe the deployment techniques we used for ThePhile.com.

Prev Next

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

The Problem

Our problem for this chapter is deploying our fully functional website to our production server.

It is a common practice to develop a site on a *development* server, then deploy to a *staging* server, and finally after a successful test on the staging server, deploy to the *production* server. We want a solution that will allow us to deploy the entire functioning site to a production server. However, we also want to be able to easily deploy the code to multiple machines so that we can test it in various scenarios. In our solution, our production server is hosted by Wrox, but it could just as easily be a segment of disk space allocated to us by a website hosting company.

So, in this chapter we want to explore the various ways we can deploy ASP.NET websites and then choose the one that best suits our needs. The chapter will provide some useful information about ASP.NET deployment that will help decide which method is most appropriate for different organizations.

Chapter 11 - Deploying the Site

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Prev Next

The Design

There are two parts to deploying the website: the database and the application. First of all, we will discuss how to deploy the database. Then we will move on to look at the web application itself.

Deploying the Data Store

A data store for a website can be anything from a set of XML files or simple Access database, to a complex SQL Server or Oracle database. Each of the website deployment options has a different set of limitations and advantages for database deployment. However, since database deployment is an important topic all on its own, we'll discuss it here rather than split up the discussion among the different installation scenarios.

Deploying a database is easiest when we own the machine to which it needs to be deployed. We can use whatever deployment scenario is most convenient for copying our particular data store. For SQL Server or Oracle there are several options, including:

- Making a backup of the development database and restoring from that backup on another machine.
- Transferring data structures and data between linked servers in some fashion, perhaps using script files.

We often don't fully control the database server. Web hosting companies often set up a single database with a certain quota of disk space, for example. In situations like this, our options are more limited. We probably can't restore from a backup, because we won't have access to Enterprise Manager against the host's database servers. Even if we do have access to Enterprise Manager, we might not have the right permissions to perform a database restore. In these cases, we are limited to using text queries to create the data structures and load the data.

For an Access database, the file just needs to be copied to a certain directory and the file is deployed. It doesn't matter what server access we have. This does pose a serious danger: if an unwanted intruder happens to find out that your Access database is available in a public internet directory, they'll be able to download it. You'll want to keep the MDB file somewhere non-obvious and preferably in a private location, so that only code from your application can access the file.

Consult your SQL Server, Oracle, or Access manual for the various options available for transferring your database from your development PC to a deployed production environment.

For automated deployment, where we create an installation program for our website, we have another option. We can take the scripts that recreate the data structures required for the application, and can have our installer execute them at install-time, guaranteeing that the data structures will be available before the application is run for the first time.

Preparing the Site for Deployment

There are three main scenarios that we will consider for deploying our ASP.NET website:

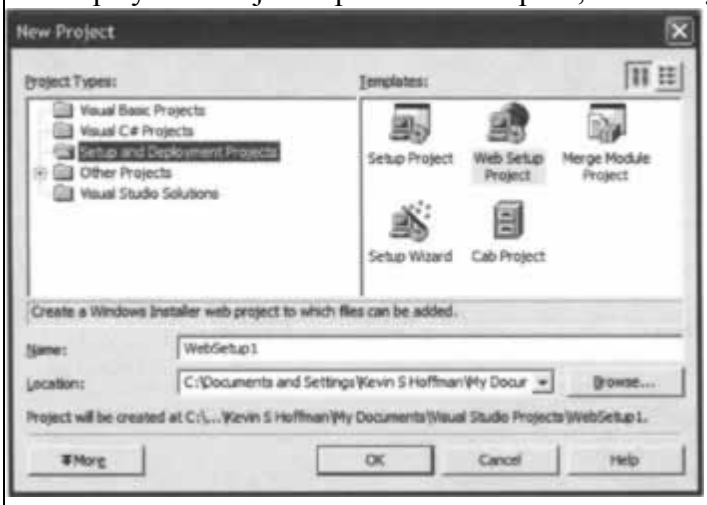
byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

The Solution

Wrox Press will host the final version of ThePhile.com, so we do not need to enlist the services of a web hosting company. This makes a deployment project in Visual Studio .NET an appropriate choice. We will create two - one with the source code and one without.

In this section we'll cover indepth the steps involved in creating a deployment project for ThePhile.com, which will result in a Windows Installer file to distribute for deploying ThePhile.

First, we open up Visual Studio .NET and choose the option to create a new project. Then we highlight the Setup and Deployment Projects option in the left pane, and the right pane shows the following options:



The following are the different types of setup and deployment projects available:

- **Setup Project -** this will create a blank installation project. It will be up to us to choose all of the various activities and files for the installer. This is typically recommended for those programmers who know ahead of time what activities need to be performed and which files need to be copied.
- **Web Setup Project -** this is similar to the blank Setup Project, except it comes with a few settings and directories already created to guide us in the right direction for installing a website.
- **Merge Module Project -** this is a slightly more advanced project for creating merge modules for use with the latest versions of the Windows Installer. Merge Modules are an advanced technique that we won't discuss here.
- **Setup Wizard -** this is basically a frontend to all of the other project types. We will choose this one - it's easy and it guides us step-by-step through creating the appropriate type of setup project.

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Summary

We have seen how to deploy and install ASP.NET websites. We've discussed some of the issues that tend to arise during deployment and the various options available for installation. After having read this chapter, you should be familiar with the following types of deployment and their associated issues and concerns:

- XCopy deployment
- Deployment to a hosted environment
- Automated deployment using VS .NET deployment projects

byMarco BellinasoandKevin Hoffman?
Wrox Press ?2002

Chapter 12: The End

This is the end of the book but it is not the end of the road. By now you should have played with the site, both by browsing the code in Visual Studio .NET, and playing with the site on your local machine.

You should also have seen how and why we built the site the way we did, and how to use those techniques on your own website.

Get Building

The next step is to start building your site. This book will have given you a framework, and some modules to use or modify. Now you will want to tailor our modules and pages to fit the needs of your site.

We also hope that you will build your own modules in our framework. A lot of our design went into making it easy to add new modules - we don't want it to go to waste! You will be able to link your modules to our central accounts system, modify our header and footer controls, and so on. This book will provide a reference for building your own modules employing similar techniques.

Chapter 12 - The End

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Join Our Community

We don't want you to develop alone. This book will have its own forum at p2p.wrox.com where you can discuss it with us and other readers. This is a great place to get help with problems, share ideas, and find out if other people have written the module you need. Or you can just show off the sites you've developed! This service is free to all readers.

Through P2P, we hope to build up a list of the best websites built with the help of this book. If you do anything really impressive, we might even ask you to write a book about it!

Read More

This book has touched on a large number of subjects: web services, server controls, security, ADO.NET, and more. If you want to find more about any of these subjects, there are several Wrox books that will help.

Web Services

This book has presented a couple of very simple web services. *Professional ASP.NET Web Services* contains lots of detailed information on web services and how to develop them. It looks at a variety of issues, including how XML is used to transmit the data, how to send complex data types, and how to ensure web services support thousands of users.

Security

Our site uses the extensibility of the ASP.NET security framework to give us a flexible accounts system. *Professional ASP.NET Security* delves deeper into these topics, showing how we can build our own custom security frameworks. It also contains many tips on ensuring code is secure.

ADO.NET

Data access and manipulation have played a major part in developing our website. Efficient use of databases is one of the best things we can do to ensure performance and scalability. *Professional ADO.NET* covers a wide range of data handling techniques, while *Professional SQL Server 2000* gives information on setting up and using SQL Server in the most effective way.

Professional Server Controls

Our site uses server controls in a number of places. Building good controls puts us in an excellent position to reuse functionality and save development effort. *Professional ASP.NET Server Controls* looks at how to build solid, reusable and flexible controls for your ASP.NET projects.

We hope that you have enjoyed this book, and that it will prove useful as you develop ASP.NET websites.

Index

A Guide to the Index

The index is arranged hierarchically, in alphabetical order, with symbols preceding the letter A. Most second-level entries and many third-level entries also occur as first-level entries. This is to ensure that users will find the information they require however they choose to search for it.

Symbols

Index

A

Index

B

Index

C

Index

D

Index

E

Index

F

Index

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Index

G

Index

H

Index

I

Index

J

Index

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Index

L

Index

M

Index

N

Index

O

Index

P

Index

Q

Index

R

Index

S

Index

T

Index

U

Index

V

Index

W

Index

byMarco BellinasoandKevin Hoffman?

Wrox Press ?2002

Prev

Index

X

Prev

ASP.NET Website Programming, C# Edition: Problem, Design, Solution

byMarco BellinasoandKevin Hoffman

ISBN:0764543776

Wrox Press 2002 (538 pages)

This book shows you how to build an interactive website from design to deployment. Packed with solutions to website programming problems, it will have you building well-engineered, extendable ASP.NET websites quickly and easily.

Table of Contents Back Cover

Back Cover

ASP.NET Website Programming shows you how to build an interactive website from design to deployment. Packed with solutions to website programming problems, this book will have you building well-engineered, extendable ASP.NET websites quickly and easily.

With *ASP.NET Website Programming* you will learn to:

- Provide flexible user accounts integrating with ASP.NET?s built-in security
- Create fully featured discussion forums
- Generate revenue from advertising
- Build a web interface for managing the files on your site
- Add opinion polls, email newsletter, and news management
- Deploy the finished site on a live server
- Build modular websites using good, n-tier coding techniques

The book?s P2P forum is a platform for exchanging code and ideas, helping to extend the website with new modules and modifications.

This book is for developers who:

- Use ASP.NET and C#
- Use Visual Studio .NET Professional or above, or Visual C#, .NET Standard
- Want to build content-based websites

About the Authors

Marco Bellinaso is a freelance software developer. He has been working with VB, C/C++, ASP and other Microsoft tools for several years, specializing in User Interface, API, ActiveX/COM design and programming. He is now spending all his time on the .NET Framework, using C# and VB .NET. He is particularly interested in e-commerce design and implementation solutions with SQL Server, ASP.NET and web services. Marco recently co-authored *Beginning C#* from Wrox Press, and is also a contributing editor for two leading Italian programming magazines.

Kevin Hoffman started working as a programmer while still in college, writing computer interfaces to solar measurement devices and various other scientific instruments. Moving to Oregon, he did everything from technical support to tuning Unix kernels, and eventually working as an ASP programmer for 800.COM, a popular online electronics retailer. From there he moved on to working on large, enterprise ASP applications. Then he finally found