Team LiB

ASP.NET Website Programming, Visual Basic .NET Edition: Problem, Design, Solution

by Marco Bellinaso and Kevin Hoffman ISBN:0764543865

Wrox Press © 2003 (544 pages)

This guide 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.

Table of Contents

Team LiB
Team LiB

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.

This book is for developers who use ASP.NET and Visual Basic .NET or use Visual Studio .NET Professional or above or Visual Basic .NET Standard.

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

- Establish a solid, scalable website foundation
- Provide flexible user accounts by integrating with ASP.NET&"CENTER"> **About the Authors**

Marco Bellinaso works as a software developer and trainer for Code Architects Srl, an Italian company that specializes in .NET. He has been working with VB, C/C++, ASP, and other Microsoft tools for several years, specializing in User Interface, API and Active X/COM design and

programming, but is now spending all his time on the .NET Framework, using C# and VB.NET.

He's been working with the .NET Framework since the Beta 1, and is now particularly interested in commerce design and implementation solutions with SQL Server, ASP.NET, and web services, with both C# and VB.NET. He is part of the VB-2-The-Max team, a popular website for VB and .NET developers, for which he writes articles and commercial software, such as add-ins for award winning VB Maximizer VB6 add-in. Marco recently co-authored *ASP.NET Website Programming, Fast Track ASP.NET*, and *Visual C#: A Guide for VB6 Developers*, all from Wrox Press, and is also a contributing editor for two leading Italian magazines: *Computer Programming* and *Visual Basic Journal.*

Kevin Hoffman started working as a programmer while still in college, writing computer interfaces to solar measurement devices and various other scientific instruments. He has done 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 .NET, which he now spends 100% of his programming and learning efforts on. Kevin has been writing on .NET for Wrox since the middle of Beta 1.

Team LiB

ASP.NET Website Programming, Visual Basic .NET—Problem, Design, Solution

Marco Bellinaso
Kevin Hoffman

Visual Basic .NET Code Provided by
Demian Martinez and Norbert Martinez

Wrox Press Ltd. *(*

© 2002 Wrox Press

All rights reserved. No part of this book may be reproduced, stored in a retrieval system or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical articles or reviews.

The authors and publisher have made every effort in the preparation of this book to ensure the accuracy of the information. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, Wrox Press, nor its dealers or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

First Printed in December 2002

Published by Wrox Press Ltd,
Arden House, 1102 Warwick Road, Acocks Green,
Birmingham, B27 6BH, UK
Printed in the United States

ISBN 1-86100-816-3

Trademark Acknowledgments

Wrox has endeavored to provide trademark information about all the companies and products mentioned in this book by the appropriate use of capitals. However, Wrox cannot guarantee the accuracy of this information.

Credits

Authors
Marco Bellinaso
Kevin Hoffman

Commissioning Editor
Dan Kent

Technical Editors
Dan Kent
Dianne Arrow
David Barnes

Index
Andrew Criddle

Project Manager
Charlotte Smith

Managing Editor
Louay Fatoohi

Visual Basic .NET Code
Demian Martinez
Norbert Martinez

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

Cover
Natalie O'Donnell

Production Coordinator
Sarah Hall

Proofreader
Chris Smith

About the Authors

Marco Bellinaso

Marco Bellinaso lives in a small town close to Venice, Italy. He works as a software developer and trainer for Code Architects Srl (www.codearchitects.com), an Italian company that specializes in .NET. He has been

working with VB, C/C++, ASP, and other Microsoft tools for several years, specializing in User Interface, API, and ActiveX/COM design and programming, but is now spending all his time on the .NET Framework, using C# and VB.NET.

He's been working with the .NET Framework since the Beta 1, and is now particularly interested in e-commerce design and implementation solutions with SQL Server, ASP.NET, and web services, with both C# and VB.NET. He is part of the VB-2-The-Max team (www.vb2themax.com), a popular website for VB and .NET developers, for which he writes articles and commercial software, such as add-ins for MS Visual Studio and other utilities for VB and .NET developers. In particular, he co-authored the award-winning VB Maximizer VB6 add-in.

Marco also loves writing and talking about technical stuff. He recently co-authored *ASP.NET Website Programming – C# edition*, *Fast Track ASP.NET* and *Visual C#: a Guide for VB6 developers*, all 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 his 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.

About the Code Converters

Usually, Wrox books include biographies and dedications for the authors only. In this case, it was decided that the contribution that the code converters had made to this edition was sufficient that they should be included in this section.

The authors of this book originally wrote all of the code in C#. Demian and Norbert did a great job of converting all of the code to Visual Basic .NET for this edition. The careful presentation and informative commenting in the code show the dedication that they had to creating a useful resource for ASP.NET developers who use VB.NET.

Demian Martinez & **Norbert Martinez**

Demian and Norbert have been messing with computers ever since their parents, also computer engineers, taught them to play games on their Apple II. In spite of being aeronautical engineers by degree, they have devoted most of their careers to computing. Their combined working experience in companies such as Gartner, PriceWaterhouseCoopers Consulting, British Airways, and Airbus has given them the opportunity to understand all phases of the software development lifecycle, from analysis to end-user training, through design, construction, and implementation.

Demian and Norbert are currently working in No Red Inc., a small technology consulting company they co-founded in Barcelona, Spain. They specialize in the development of web applications for small- to medium-sized companies using ASP.NET, VB.NET, and C#.

Acknowledgements

Demian would like to thank his wife Mary for her support, her understanding, and her amazing smile. Norbert would like to thank his partner Iratxe for her support, her love, and for not being too jealous when he spends some time with the computer. Both Demian and Norbert would also like to thank their parents for being so special (and letting them play with the Apple II).

Team LiB
Team LiB

Introduction

Overview

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 from 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.

Team LiB
Team LiB

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 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.

Chapter 1 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.

Chapter 2 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.

Chapter 3 extends our foundations to the presentation layer. We build base classes for the ASP.NET pages in the site, a custom error page, and site-wide navigation, header, and footer controls.

Chapter 4 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.

Chapter 5 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.

Chapter 6 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.

Chapter 7 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.

Chapter 8 covers opinion polls and voting. We look at how to administer questions, log votes, and collate them into useful reports.

Chapter 9 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.

Chapter 10 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.

Chapter 11 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.

Chapter 12 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.

Team LiB
Team LiB

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 Visual Basic .NET*, or *Professional ASP.NET* and a VB.NET book.

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

Team LiB
Team LiB

What You Need to Use This Book

To run the code 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 Basic .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.

Team LiB
Team LiB

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 styles. 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?>
<Invoice>
   <part>
      <name>Widget</name>
      <price>$10.00</price>
   </part>
</invoice>
```

In cases like this, the code with a white background is code we are already familiar with; the line highlighted in gray is a new addition to the code since we last looked at it.

Advice, hints, and background information comes in this type of font.

Important Important pieces of information come in boxes like this.

Bullets appear indented, with each new bullet marked as follows:

- **Important Words** are in a bold type font.
- Words that appear on the screen, or in menus like File or Window, are in a similar font to the one you would see on a Windows desktop.
- Keys that you press on the keyboard, such as *Ctrl* and *Enter*, are in italics.

Team LiB
Team LiB

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 Visual Basic .NET Edition*. 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 Visual Basic .NET Edition*. 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** (8163), and **page number** of the problem in the Subject field.
- Your **name**, **contact information**, and the **problem** in the body of the message.

We *won't* send you junk mail. We need the details to save your time and ours. When you send an e-mail message, it will go through the following chain of support:

- Customer Support - Your message is delivered to our customer support staff, who are the first people to read it. They have files on most frequently asked questions and will answer anything general about the book or the website immediately.
- Editorial - Deeper queries are forwarded to the technical editor responsible for that book. They have experience with the programming language or particular product, and are able to answer detailed technical questions on the subject.
- The Authors - If even the editor cannot answer your problem, they will forward the request to the author. We do try to protect the author from any distractions to their writing, but we are happy to forward specific requests to them. All Wrox authors help with the support on their books. They will e-mail the customer and the editor with their response, and again all readers should benefit.

The Wrox Support process can only offer support to issues that directly relate to the content of the book. Support for questions that fall outside the scope of normal book support is provided via the community lists of

our http://p2p.wrox.com/ forum.

p2p.wrox.com

For author and peer discussion join the P2P mailing lists. Our unique system provides **programmer to programmer™** contact on mailing lists, forums, and newsgroups, all in addition to our one-to-one e-mail support system. If you post a query to P2P, you can be confident that the many Wrox authors and industry experts who use our mailing lists will examine it. At p2p.wrox.com you will find a number of different lists that will help you, not only while you read this book, but also as you develop your own applications.

This book has its own list called aspdotnet_website_programming. Using this, you can talk to other people who are developing websites using the methods and framework presented here. You can share ideas and code for new and improved modules, get help with programming headaches, and show off the sites you've written!

To subscribe to a mailing list just follow these steps:

1. Go to http://p2p.wrox.com/.
2. Choose the appropriate category from the left menu bar.
3. Click on the mailing list you wish to join.
4. Follow the instructions to subscribe and fill in your e-mail address and password.
5. Reply to the confirmation e-mail you receive.
6. Use the subscription manager to join more lists and set your e-mail preferences.

Why This System Offers the Best Support

You can choose to join the mailing lists or you can receive them as a weekly digest. If you don't have the time, or facility, to receive the mailing list, then you can search our online archives. Junk and spam mails are deleted, and the unique Lyris system protects your e-mail address. Queries about joining or leaving lists, and any other general queries about lists, should be sent to listsupport@p2p.wrox.com.

Team LiB
Team LiB

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.

Team LiB
Team LiB

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.

Important 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.

Team LiB
Team LiB

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 to make that easy.

Working in modules and using controls already goes some way towards this. Particularly, using controls means that non-programmers can edit the pages of our site more easily - nearly all they see is HTML code. A control just looks like another HTML tag.

Working in modules means that new modules can be added to the site at any time, with minimum disruption. All modules are fairly independent, so new ones can be added - and changes made - pretty easily.

Each individual module needs to be easy to change. A good way to do this is to work in layers, or 'tiers'. We will be using a three-layer design for most modules. We have a data layer, a business layer, and a presentation layer. Data passes from **data layer** to **business layer**, and from business layer to **presentation layer**, and back again. Each layer has a job to do. Underneath the data layer is a data source, which it is the data layer's job to access.

The data layer obtains fairly raw data from the database (for example, "-10"). The business layer turns that data into information that makes sense from the perspective of business rules (for example, "-10 degrees centigrade"). The presentation layer turns this into something that makes sense to users (for example, "Strewth! It's freezing!").

It's useful to do this, because each layer can be modified independently. We can modify the business layer, and provided we continue to accept the same data from the data layer, and provide the same data to the presentation layer, we don't need to worry about wider implications. We can modify the presentation layer to change the look of the site, without changing the underlying business logic.

This means we can provide versions of the site for different audiences. We just need new presentation layers that call the same business objects. For example, providing different languages: "Zut alors! Comme il fait froid!", "Allora, è freddo!", and so on.

We need methods to get changes we make onto the live site. This could be through FTP uploads, but in many circumstances it is better to work through a web interface.

We will also need tools to administer the other sections - ban problem users, add news articles, and so on. This is all part of providing a maintainable site.

Community

Sites generally benefit from allowing readers to contribute. Because our site is not intended for users to spend hours looking at, our community features must not require a lot of users' time.

There are two ways that we will build our community: through polls and forums. Polls give users the opportunity to give their opinion in a single click - so they require very little time from the user, but can make a site seem far more alive.

Forums enable users to discuss topics with other users. Messages remain in the system, and replies are posted. Readers can leave a post, and then come back later to see if there are replies. This is more appropriate for our purposes than a chat room, which requires the reader to concentrate on the site for the whole duration of the chat.

Community can really give a site a life of its own. Over time, strong characters, heroes, and villains emerge. Many sites depend entirely on community, and become extremely popular - for example www.plastic.com.

For any of this to work, we need to identify users and provide them with unique logons. So our system will need some form of user accounts system.

Interesting Content

The content most relevant to our users will be movie- and-book-related news and reviews. This content tends to be highly relevant for a short period of time: after a story has broken, or immediately after a release. Our site will need tools to manage news in this way.

Another way to provide interesting content is to get somebody else to provide it! This is part of what we're doing with our community section. Part of the purpose of building community is to get people contributing content.

Advertising

Advertising generates revenue (or in some cases it is used to exchange banners with other sites). We need to display adverts, and record data about how often each advert has been displayed and clicked on.

We also need to gather information about what the users of the site like, so we can target our advertising content. Polls and forums can provide us with useful information when finding products to advertise.

The biggest sites target individual users based on their demographic and any other information gathered about them (for example, Yahoo! and Amazon.com target advertising and product recommendations to the demographic and buying habits of each user). Our site already has a fairly narrow target demographic, and is not particularly big, so we don't need to do this.

Frequent Visits

A good site will make people want to return. If the content is compelling, and there's plenty of discussion going on, then people visit again and again.

It's still a good idea to remind users from time to time. We want to draw attention back to the site, even when the user isn't viewing it. One way we'll be doing this is through an e-mail newsletter, which gives users useful information and subtly reminds them to visit the site.

We will also build a Windows application that acts as a news ticker, with automatically updating news

headlines. Users can click a headline to view the full story on the site.

Team LiB
Team LiB

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

Chapters 2 and 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.

There will be admin facilities to create adverts, select how frequently they should be displayed, and start and end campaigns.

Encouraging Community

Chapter 8 will cover our voting system, and forums will be covered in Chapter 10. The voting system will allow administrators to create new questions to vote on. Answers will be recorded and displayed, and an archive of old results maintained – accessible from a standalone Windows application. We guard against multiple votes from the same user by using cookies and IP number.

The forums system will let each user choose an avatar image to represent them, and start posting. Discussion will be organized into categories, and within them there will be various topics. Users can post new topics, and reply to existing topics. We use regular expressions to allow formatting tags in messages, but prevent images or JavaScript.

Getting Repeat Visitors

As well as providing all this great content, we will include two features specifically for getting visitors back to the site.

The first is covered in Chapter 6 where we look at news. We will develop a web service that exposes our news headlines. We will then build a Windows client that displays the headlines, updating itself regularly. Clicking a headline will open a browser on the correct page for the full story.

The second is covered in Chapter 9. We will create the facility for visitors to subscribe to receive e-mail updates from us. Once they are subscribed, we send a mail out regularly to encourage repeat visits. This mail will include highlighted news and features, and links back to the site. We will develop a system that enables administrators to create plain text and HTML messages. We then develop a mailing list admin module for creating subscription forms for new mailing lists, administering list members, adding newsletters, and managing subscriptions. Messages can include custom tags so that each list member receives an e-mail tailored to their own details.

Deploying the Site

Although we haven't mentioned it before, we will eventually need to move the site from our production machine to the live server. This can be a complex task, because we need to separate the files needed for the site to run from the source code files that we only need for development. We will look at this in Chapter 11, and see how Visual Studio .NET gives us tools to make the process easy.

Team LiB
Team LiB

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.

Important The code and database are 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.

Team LiB
Team LiB

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.

Team LiB
Team LiB

The Problem

Building a solid foundation can be a daunting task. It requires a good understanding of how the application will operate before we've gone 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 website must support. This will affect the HTML and client-side scripts that your developers can work with, and hence be part of the coding standard. As far as ThePhile.com is concerned, we will not be dictating the use of any particular browser. We will try to code the pages so that any recent browser that supports the latest HTML standards can use them.

We also need to consider what the purpose of the website is, and who will be the users. Many businesses, ranging from the small start-up business to the huge worldwide corporation, provide services and applications for their employees on their intranet. There are many different types of applications that fall into this category, including:

- HR applications - many large corporations provide systems on the web to automate many tasks for dealing with the employee's day-to-day business, such as time sheets and benefits tracking. These applications require high security and availability.
- Internal support applications - as well as creating software that is deployed to their customers, companies have various departments that often have 'in-house' software designed to support their own needs. These applications require security, reliability, availability, and often a high degree of support from the programming staff.

These types of applications have specific deployment issues, which often arise due to a wide disparity in system configuration and type across the employees requiring the software. Other deployment concerns arise simply due to the large number of employees that must make use of this software. It is also becoming more common for web application vendors to create an application, build a deployment program, place it on a CD, and then sell that CD to customers who then deploy that application throughout *their* intranet. For example, there are several companies that provide defect tracking solutions that are essentially websites you install from a CD to support your programming intranet. The possibilities are extremely wide and varied. You may not know your particular solution for deployment at the time you are defining your problem, but you should definitely be aware that it must be a core part of the design of your website foundation.

Finally, our website wouldn't look very much like a website without a user interface. So we obviously need some type of UI. Putting some effort into the design of the user interface before a lot of code has been written can have extremely large payoffs. We will need to take into consideration our audience when we design the look and feel of our website, as well as the navigation and flow of the site, to make it easy for the target audience to use and traverse.

Now that we've covered a little bit about the overall problems that face ASP.NET website architects, let's take a look at the problem statement we came up with for the foundation of our website. We had special needs for ours, because the developers of our website have never physically been in the same room.

The Problem Statement

For our purposes the problem statement includes stating the problem we are attempting to solve, and the constraints that we must conform to in solving that problem. Our problem statement is divided into two sections: a vision (or purpose) and a set of requirements. Depending on what particular software development process you use, your problem statements may vary significantly from the one we will present here. If you are a fan of the Microsoft Solutions Framework (MSF) then you might already be used to producing a vision document and a requirements document.

We'll present our vision statement and then list the requirements for our product. It is absolutely imperative that you do not start a single line of code or actual design until you have adequately defined these for your project. In many iterative processes, you may be satisfied with only partially defining the requirements, because you know you will revisit the requirements document multiple times throughout the lifetime of your project.

The Vision

We are endeavoring to build a complete, content-driven website that illustrates the importance of modular building and will hopefully illustrate a few ASP.NET 'best practices' along the way. We will develop a solid, scalable foundation on which to build the modules that will be developed throughout the rest of this book. A secondary goal is to provide a foundation that can be used by multiple programmers with diverse experience and still produce a coherent, cohesive solution.

The Requirements

It is important that we keep our requirements separate from our purpose. The requirements are the rules to which our design must conform in order to produce the solution we set out to create. In an iterative process, the requirements generally change with each iteration. In our small development environment, we won't need an iterative process, so the following is the list of requirements that we defined for our project:

- **Scalability** - our solution must be scalable. It must be able to expand to meet increasing performance demands with a minimum of extra coding required. It's a lofty goal, but it is quite possible with the right design.
- **Flexibility** - our solution must be *agile*. This may be a buzzword, but there is some validity behind it. We must try to make the foundation of our website agile in such a way that changes encountered during the development of a module that require modification of the foundation will not drastically affect the rest of the site.
- **Reusability** - our solution for the core foundation of our website must be designed in such a way that it promotes code reuse. A strong emphasis should be placed on object hierarchies, inheritance, and reuse, starting with the foundation and carrying on through all of the modules in the website.
- **Separation** - our core foundation code should provide a solid foundation for the rest of the website, but it should not be so closely tied to it that changes to individual modules will have an impact on the core foundation code.
- **Deployment** - our application should be coded in such a way that it can be deployed on the Internet for public use, and also to workstations running Windows 2000 and XP to allow programmers to examine and learn from the source code.
- **Test plan** - as experienced programmers we know that developing a large project, even one that may appear simple on the outside, is going to be a difficult process. As such, we need to make sure that we have an organized way in which we test our code so that we can be reasonably confident that there are no bugs in it when it is released to production.

In summary, the foundation for our website, ThePhile.com, needs to provide a stable, solid, scalable foundation that will give us the flexibility to make changes throughout the development process and later, as well as provide enough standardization and convention to allow a team of programmers to build portions of the website separately, allowing for easy integration of individual modules.

Team LiB
Team LiB

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 provide.

Once again, we're faced with finding a compromise that benefits everybody. Standardization not only allows teams of programmers to produce code that follows the same conventions, making it easier to read and maintain, but it allows for the same programmer to write consistent code. Far more often than we like to admit, programmers will use one convention one day, and another convention the next. Without some sense of enforced structure to the programming, the infamous spaghetti code will rear its ugly head and make life miserable for everyone involved. Another common practice that ensures solid, standardized code is the use of **code reviews**. Code reviews are where other programmers (or managers, depending on skill distribution) review their peers' code for accuracy, efficiency, and compliance to coding standards and conventions. Some programmers resist this kind of practice, but it can be extremely valuable and productive.

The guidelines here tend to match the recommendations that Microsoft issues to its own .NET development teams. If we haven't pointed out a difference between our standards and Microsoft's, then they're essentially the same. When in doubt, it is generally a good idea to favor established methods that developers are already familiar with. Change can often bring with it benefits; however, it can also be a thing that programmers resist strongly.

Naming Guidelines

Naming guidelines actually cover two things: **naming** and **casing**. The following is a list of generic guidelines that apply to both naming and casing. Microsoft strongly recommends the use of a capitalization scheme called **Pascal casing**. Pascal casing is a scheme where all words in an identifier have the first letter capitalized and there is no separation character between words. Another type of capitalization scheme is called **camel casing**. This is where the first letter of the identifier is lowercased, and thereafter the first letter of each word is capitalized. The following table is a summary of Microsoft's capitalization suggestions:

Type	Case	Additional Information
Class	PascalCase	Examples: *MyClass*, *Utility*, *DataHelper*
Enum value	PascalCase	Examples: *Colors.Red*, *PossibleValues.ValueOff*
Enum type	PascalCase	Examples: *Colors*, *PossibleValues*
Event	PascalCase	Examples: *MouseClick*, *ButtonDown*
Exception class	PascalCase	Class name ends with Exception suffix, for example: *MyCustomException*, *WebServiceException*
Interface	PascalCase	Interface name is prefixed with the letter I, for example: *ICar*, *ISerializable*
Method	PascalCase	Examples: *GetItemData*, *UpdateModifiedValue*
Namespace	PascalCase	Examples: *Company.NewApplication.DataTier*
Property	PascalCase	Examples: *ItemValue*
Parameter	camelCase	Examples: *itemArray*, *valueData*, *purchasePrice*
Private member variable	camelCase	Microsoft makes no recommendation on this; however, it is useful to distinguish private member variables from other identifiers

In addition to the above summary of capitalization rules, the following guidelines apply to naming classes, interfaces, and namespaces:

- Do *not* use class names that overlap with namespaces, especially those namespaces that are supplied by Microsoft. So stay away from naming your classes things like System, UI, Collections, or Forms.
- Do *not* use the underscore character. Many of us who have been writing C++ code for a long time have developed the habit of using a preceding underscore to indicate a private member variable within a class. This practice has fallen from grace, and is now discouraged.
- Do *not* use identifier names that conflict with keywords. Most languages won't let you anyway!
- Do *not* use abbreviations in your identifiers. Also, where you use an acronym, treat it as a word – don't use all uppercase. For example, the .NET Framework has namespaces such as SqlClient (not SQLClient).
- *Do* follow the casing conventions in brand names. For example, if you place your company name in a namespace, and your company name has a specifically branded capitalization scheme (for example NeXT or IBM, both of which have a capitalization scheme that doesn't coincide with the casing recommendations) you should retain your company's branding. So, you would not reduce IBM in a namespace to Ibm, nor would you reduce NeXT in a namespace to Next.
- *Do* use nouns and noun phrases when naming your classes and namespaces. This is highly recommended and preferred over using verbs. For example, use Parser as a namespace or class name, rather than Parse or Parsing. Verbs should be used for method names only
- Do *not* use Hungarian notation when naming things. For example, in classic VB, controls were often given names like btnConfirm, which would immediately tell the reader that it was a button. Microsoft's style guidelines are now recommending that people do not prefix their variable names with anything related to that variable's data type. Microsoft feels that the development tools (specifically VS.NET) should provide information pertaining to a given member's data type by such means as intelligent hovering pop-up dialogs. A better purpose for a variable name is to describe its use rather than its data type. Interestingly enough, Microsoft *does* recommend the usage of Hungarian