



F R O M T E C H N O L O G I E S T O S O L U T I O N S

Building Websites with the

ASP.NET Community Starter Kit

A practical guide to understanding, implementing, and extending this powerful and free application from Microsoft

K. Scott Allen
Cristian Darie

[PACKT]
PUBLISHING

Building Websites with the ASP.NET Community Starter Kit

K. Scott Allen
Cristian Darie



Building Websites with the ASP.NET Community Starter Kit

Copyright © 2004 Packt Publishing

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 embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, Packt Publishing, nor its dealers or distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

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

First edition: May 2004

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 1-904811-00-0

www.packtpub.com

Cover Design by www.visionwt.com

Credits

Authors

K. Scott Allen
Cristian Darie

Technical Editors

Deepa Aswani
Saurabh Bijur
Dipali Chittar

Technical Reviewers

Ovidiu Platon
Dave Ranck
Colin Mumford

Indexer

Vijay Tase

Commissioning Editor

Douglas Paterson

Proofreaders

Chris Smith
Vijay Tase

Layout

Niranjan Jahagirdar

Cover Designer

Helen Wood

About the Authors

Cristian Darie is an experienced programmer specializing in various Microsoft *and* open source technologies, and relational database management systems. Having worked with computers since he was old enough to press the keyboard, he initially tasted programming success with a first prize in his first programming contest at the age of 12, and from there moved on to many other similar achievements in years that followed. As an eagle-eyed technical reviewer with Wrox Press, Cristian was tempted into writing and has since co-authored several .NET and database books for Wrox Press and Apress, including *Beginning ASP.NET E-Commerce* and *The Programmer's Guide to SQL*.

Cristian is currently building a large community website using the Community Starter Kit for University "Politehnica" of Bucharest. He can be contacted through his personal website at <http://www.CristianDarie.ro>.

K. Scott Allen is the Chief Software Architect for Medisolv, Inc., and holds MCP and MCSD certifications. In 12 years of software development, Scott has worked on everything from real-time firmware in 8-bit embedded devices to highly scalable Internet applications connecting COM+ and J2EE components with web services. Living in Hagerstown, Maryland, Scott tries to play as much softball as possible during the summer. Scott is a cofounder of the site www.odeToCode.com, a .NET developer resource built using the Community Starter Kit. You can reach Scott at scott@odeToCode.com.

I would like to thank Douglas Paterson and Poonam Lall for their generous help and support while working on this book.

Table of Contents

Chapter 1: The ASP.NET Community Starter Kit	1
What Can the Community Starter Kit Do for You?	2
The License Agreement	3
Installing the Community Starter Kit	3
Summary	9
Chapter 2: Touring a Community Starter Kit Website	11
Overview of the Community Starter Kit	11
The Main Page	12
The Community Logo	13
The Community Menu	13
The Search Box	13
Login/Register	14
Advertising	14
Web Boxes	14
Web Services Boxes	14
Voting Polls	14
The Contents	15
Overview of the CSK Modules	15
Articles	16
Events	18
Books	19
Links	20
Discussions	21
Photo Gallery	23
Downloads	24
Parent Section	25
HTML Page	26
Common Features in the CSK Modules	26
Security	26
Moderation	27

Table of Contents

Comments	28
Ratings	30
Notifications	30
Quotas	31
HTML Editor	32
Creating and Using User Accounts	33
Creating a New User Account	33
Log In	34
Edit Profile	34
Logging in as Administrator	34
Summary	35
 Chapter 3: Creating and Administering Community Websites	 37
Using the Community Administration Menu	37
Edit Sections	39
Editing a Section	41
General	41
Appearance	42
Web Boxes	43
Security	44
Web Services	45
Advanced	47
Add New Section	51
Edit Voting Polls	51
Edit Service Subscriptions	53
Edit Messages	56
Edit Topics	58
Edit Users	59
Edit Web Box Services	60
Send Newsletter	62
Edit Images	64
View Activity Log	65
Delete Content	66
View Documentation	66
Administering Advertisements	66
Administering Quotes	67
The ISP Administration Page	68

Creating a New Community	69
Summary	73
Chapter 4: Themes, Skins, and the Common Controls	75
Meeting Themes, Skins, and Styles	75
Playing with Visual Studio	77
Painting the Skin with Style	78
What are Skins Made Of?	79
Page Skins	79
Content Skins	81
Control Skins	84
Template Skins	85
The Web Boxes	86
Creating Themes	87
The Community Starter Kit Custom Controls	88
Learning about the Common Content Controls	89
AddContent	89
Author	91
BriefDescription	92
CommunityImageButton	93
ContentList	93
DateCreated	95
DisplayQuota	96
HtmlTextBox	98
ItemAuthor	98
ItemBriefDescription	99
ItemDateCreated	99
ItemEditContent	99
ItemHasRead	99
ItemImage	100
ItemReadMoreLink	100
ItemTitle	101
ItemTitleLink	101
ItemViewCount	102
LengthValidator	102
Logo	103
PageFooter	103
PageHeader	103
SectionTitle	104
Sorter	105
Title	106
UploadValidator	106
ViewCount	107

Summary	108
Chapter 5: Customizing your Website	109
Setting up a New Community	111
Preparing the Folder Structure and the Page Skin	111
Adding Information	113
Having a New Home	113
About Me	115
Advertising the Books	117
Forum	119
Changing the Layout	120
Updating the Parent Section and the Styles	120
Summary	124
Chapter 6: The Community Starter Kit Core Architecture	125
The CSK Folder Structure and Configuration Files	125
Configuring the CSK Project	126
Web.Config	126
ISP Administrator Information	127
Default Page Information	127
Database Connection Information	127
Enabling Web Services Timer	127
Enabling Debug Mode	128
Handling Client Requests	128
Configuring Authentication Information	128
Setting the Maximum Request Length	128
Handling Image File Requests	128
Configuring Web Services	129
Global.asax	129
What Do the Main CSK Folders Contain?	129
The Admin Folder	129
The Advertisements Folder	131
The Communities Folder	131
The ISPAdmin Folder	132
The Engine Folder	134

Building Pages for the Client	134
The Default Page	134
Analyzing the Community Default Page	135
Processing Client Requests	139
Processing the Request in Application_BeginRequest	139
Getting Page Information in GetPageInfo	141
Authenticating the Request in Application_AuthenticateRequest	143
Processing the Request in the Middle Tier	145
The Community Starter Kit Framework	146
The Framework Base Classes	146
The CommunitiesSectionHandler Class	147
The CommunityGlobals Class	147
The SkinnedCommunityControl Class	147
ContentAddPage, ContentEditPage, ContentItemPage, and ContentListPage	147
The PageInfo Class	149
The QuotaInfo Class	150
Handling Communities	150
Handling Sections	152
Handling Images	155
Handling Content Pages	156
Handling Search	158
Studying the Engine/Admin Folder	158
Modules for the CSK	159
The Database Structure	161
The Tables	161
Online CSK Resources	164
Summary	164
Chapter 7: Inside a CSK Module	165
Database Layer	166
Procedures and Functions	168
Community_ArticlesAddArticle	168
Community_ArticlesGetArticles	169
Ordering Content	169
Paging and Sorting Content	171
Community_ArticlesGetArticle	173
Delete Content	174

Module Classes	174
Components	175
Component Helpers	177
Get Articles	177
Content	179
communityDefault.aspx	180
Article Content Control	181
The Article Skin	188
InitializeSkin Method	190
Skins and Styles	191
Summary	192
 Chapter 8: Creating a New Module	 193
Module Design	193
Module Construction Blueprint	196
The Community_Faq Table	197
The Community_Faqs Stored Procedures	198
Community_FaqsAddFaq	198
Community_FaqsEditFaq	199
Community_FaqsGetFaqs	200
Community_FaqsGetFaq	201
Initializing the FAQ Module	202
FAQ Components	203
FaqInfo	203
FaqUtility	204
FAQ WebControls	206
FaqAnswer and FaqReference	206
FaqEditContent	208
Content Classes	208
Faq and FaqSection	209
AddFaq and EditFaq	209
FAQ Page Content Skins	212
Module Styles	215
Putting It Together	215
Summary	216

Chapter 9: Customization with User Controls	217
The Custom Module Section	217
The "Recent Logins" User Control	217
Community_RecentLogins Stored Procedure	218
Recent Logins.ascx	218
Recent Logins Administration	221
Custom Web Boxes	223
FAQ Web Boxes	224
Current Event Web Box	229
Summary	236
Chapter 10: Adding an RSS Feed	237
The RSS 2.0 Specification	237
Community RSS Design	238
HttpModule Changes and Configuration	239
CommunityRSS.aspx	240
Testing the RSS Feed	243
Advertising the RSS Feed	245
Summary	246
Chapter 11: Deployment	247
Installing Content Files	247
Copy Project	248
Xcopy Deployment	248
Configuring ASP.NET	250
ASP.NET on Windows 2000	250
ASP.NET on Windows 2003	250
CSK Site Configuration	251
Database Deployment	254
DTS Deployment	254
Script Deployment	256
Backup and Restore	257
Database Ownership Problems	257
Community Starter Kit Configuration	258
Web.Config	258

Table of Contents

ISPAdmin	258
Working with Other Sites	259
Summary	259
Appendix A: CSK Controls	261
BriefDescription	261
ContentList	261
DateCreated	262
DisplayQuota	262
HtmlTextBox	262
ItemReadMoreLink	263
ItemTitleLink	263
LengthValidator	263
PageFooter	264
SectionTitle	264
Sorter	265
Title	265
UploadValidator	265
Index	267

Introduction

The ASP.NET Community Starter Kit allows you to quickly create a fully featured community-driven website, complete with article and news management, downloads, forums, and user handling. Available for free from Microsoft, this application puts a staggering range of functionality into your hands, and either by using it as is, or customizing it to your requirements, you are giving your projects a great head start.

Supported and tested by thousands of developers in the ASP.NET community across the world, the Community Starter Kit offers you the luxury of a scalable and extensible architecture, as well as the ability to control the look and feel of your site.

The book is structured to help you understand, implement, and extend the Community Starter Kit; it will take you inside the Community Starter Kit, allowing you to harness its power for easily creating your own websites.

What This Book Covers

Chapter 1 introduces the Community Starter Kit (CSK), and shows how to install it so you are up and running. *Chapter 2* takes us on a tour of the sample community website that ships with the CSK—the Lunar .NET community. This chapter is your orientation to the features and functionality provided by the CSK, and is our first step towards creating our own community website.

In *Chapter 3* we see how to administer the data that gets displayed by the CSK, and learn how to manage our community, and multiple communities on the same installation, through the various administration tools provided by the CSK.

The Community Starter Kit has been designed from the ground up to support easy integration of skins and styles, allowing you to change the look and feel of your site with ease. In *Chapter 4* we take a detailed look at themes, skins and the common controls that control the overall appearance of your site. Before we delve into the CSK framework and architecture, we have a quick walkthrough of implementing a completely new CSK site in *Chapter 5*.

Understanding the core architecture of CSK is essential if you want to extend the system or even modify the existing code. In *Chapter 6* we'll learn how the Community Starter Kit builds the client pages, and what are the major classes that drive it.

In *Chapter 7* we look inside one of the modules that ship with the CSK to gain an overall view of the structure of a CSK module.

With knowledge of how to work with the administrative features and an understanding of the inner workings of the CSK, in *Chapter 8* we move on to extending the CSK by creating a new module: a Frequently Asked Questions module. The strategies and principles learned in this chapter will prepare you for developing custom modules to add your own functionality to the CSK.

Creating a new module from scratch is not the only way to customize the CSK application, and in *Chapter 9*, we look at customizing through the use of user control (.ASCX) files. The flexibility of the CSK is further demonstrated in *Chapter 10*, where we build an RSS feed to syndicate the output from any part of the site.

The final chapter of the book, *Chapter 11*, shows how to deploy a CSK website to a production environment, and tackles preparing the web server, and deploying both the code and the database to your hosting setup.

What You Need for Using This Book

This book has been written for ASP.NET developers with a sound grasp of C#. To use this book you need to have the following:

- Access to Visual Studio .NET Professional or higher

This book uses the C#, Visual Studio. NET version of the ASP.NET Community Starter Kit available from:

<http://www.asp.net/StarterKits/DownloadCommunity.aspx?tabindex=0&tabid=1>

To install and run the ASP.NET Community Starter Kit, you will need the following:

- The .NET Framework
- One of Windows Server 2003, Windows 2000, or Windows XP operating systems
- An installation of SQL Server 2000 or MSDE

Conventions

In this book you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

There are three styles for code. Code words in text are shown as follows: "The `GetResponseInfo` method call will retrieve enough information to populate the `ServiceResponseInfo` object".

If we have a block of code, it will be set as follows:

```
private void GetResponseInfo()
{
    responseInfo = new ServiceResponseInfo();
    if(!sectionInfo.IsSectionWebService)
    {
        responseInfo.ServiceTitle = "Unpublished";
        responseInfo.ServiceDescription =
            "Section is not published as a service.";
    }
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines will be made bold:

```
if (Context != null)
{
    SectionInfo objSectionInfo = (SectionInfo)Context.Items["SectionInfo"];
    _text = objSectionInfo.Title;
```

```
if(objSectionInfo.IsSectionWebService)
{
    _text += String.Format(
        "&nbsp; <a href='{0}'><img src='{1}'" +
        "alt='RSS Feed'></a>",
        "communityRSS.ASPX",
        CommunityGlobals.AppPath +
        "/communities/common/images/xml.gif");
}
```

New terms and **important words** are introduced in a bold-type font. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "clicking the Next button moves you to the next screen".

Tips, suggestions, or important notes appear in a box like this.

Any command-line input and output is written as follows:

```
>xcopy /s *.* ..\stage
cd ..\stage
```

Reader Feedback

Feedback from our readers is always welcome. Let us know what you think about this book, what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an e-mail to feedback@packtpub.com, making sure to mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, then please send us a note in the Suggest a title form on www.packtpub.com or e-mail suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, then see our author guide on www.packtpub.com/authors.

Customer Support

Now that you are the proud owner of a Packt book, we have a number of things for you to get the most from your purchase.

Downloading the Example Code for the Book

Visit <http://www.packtpub.com/support>, and select this book from the list of titles to download any example code or extra resources for this book. The files available for download will then be displayed.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or a code error—we would be grateful if you could report this to us. By doing this you can save other readers from frustration, and also help to improve subsequent versions of this book.

If you find any errata, you can report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the Submit Errata link, and entering the details of your errata. Once your errata have been verified, your submission will be accepted and the errata added to the list of existing errata. The existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Questions

You can contact us at questions@packtpub.com if you are having a problem with some aspect of the book and we will do our best to address it.

1

The ASP.NET Community Starter Kit

Everybody agrees that .NET brought in many improvements over previous development paradigms and technologies. The drawback with new technologies is that developers need time to master the new skills; the large code bases, solutions, techniques, and expertise that were available with older technologies are no longer applicable within the framework of new technologies. It is always more difficult to build a complex solution with a technology you haven't yet mastered, no matter what that technology is.

In order to help developers become more efficient with ASP.NET, Microsoft has released a number of starter kits that help ASP.NET developers to create their own solutions in a faster and easier manner.

The ASP.NET Starter Kits are complete solutions that you can update and customize for your own needs. When building complex applications, designing the application architecture is often a time-consuming step. Borrowing the entire architecture from an existing solution can be of immense help if the job is done right.

At the time of writing, there were five starter kits available for download at <http://www.asp.net/starterkits> (visit the site for more information on these):

- Time Tracker Starter Kit
- Reports Starter Kit
- Community Starter Kit
- Commerce Starter Kit
- Portal Starter Kit

The names of the starter kits are self-describing.

This book is exclusively dedicated to the **Community Starter Kit (CSK)**, which is the most complex and powerful of all the starter kits. It provides a complete application that can be adjusted exclusively using the administrative interface included within the starter kit without touching a single line of code or knowing its internal working.

In the first five chapters, you'll learn how to create your own customized solution by using the administrative interface that comes with the Community Starter Kit application. In Chapters 6 through 11, you will learn how to programmatically create new modules and plug them into the site.

What Can the Community Starter Kit Do for You?

The CSK helps you to create and administer online community websites by providing the complete source code for a carefully tested, scalable, and modular structure that can be reused and customized to suit your particular needs.

Community websites are websites that support online communities. These can be regional user communities, developer resource sites, news sites, or other kinds of user-group sites. Most community websites provide features such as presenting news and events concerning the community, photo galleries, links to other websites, discussion forums, and so on.

The CSK application itself is quite complex but can save you many hours of design and development efforts. It is built using powerful and flexible 3-tier architecture with reusable components and offers good performance. We'll discuss the details of the architecture starting with Chapter 6, but an example of its modularity is that the application, excluding the administration pages, has only two ASPX web forms. Each client page is dynamically built using separate web user controls and data gathered from the SQL Server database.

By default, the CSK comes with out-of-the box functionality that includes support for nine types of content including articles, books, events, photo galleries, downloads, user polls, and more. It also supports features such as moderation, upload quotas, comments, user polls, ratings, newsletters, advertisements, web services, and security. We'll take a detailed look at each of these in Chapters 2 and 3.

To see how these features work, you can preview the Lunar .NET community (the sample community that ships with the CSK) at <http://www.asp.net/CommunityStarterKit/>.

The CSK has built-in functionality for **skinnable** interfaces, which means you can change its appearance entirely through its administration interface and configuration files without modifying a single line of code.

An advantage of using the CSK is that a number of Internet Service Providers (ISPs) support automatic deployment of websites that are based on it. We'll talk more about deployment techniques and strategies in Chapter 11.

The License Agreement

The license agreement for the ASP.NET Starter Kits specifies that apart from some restrictions, you're allowed to use them for any commercial or non-commercial purposes. Thus, you're allowed to create your own websites based on the CSK code and use them for your own needs.

The complete license agreement terms for using the starter kit application can be found at <http://www.asp.net/samplesource/license>. Even though the terms are pretty relaxed, please read them carefully before starting to create your websites based on the CSK.

Installing the Community Starter Kit

Before moving on to the installation details, make sure you have met the necessary system requirements:

- .NET Framework version 1.0 or 1.1
- Windows 2000, Windows XP Professional Edition, or Windows Server 2003
- SQL Server 2000 or MSDE 2000
- Internet Information Services (IIS)

You can't use the CSK with Windows XP Home Edition because it doesn't include IIS.

Before installing the starter kit, make sure you have the appropriate software and hardware environment. If you don't have SQL Server, you can install it from the Visual Studio .NET kit or from its official homepage, <http://www.microsoft.com/sql/msde/>. You can install IIS by going to the Windows Components Wizard located at Control Panel | Add/Remove Programs | Add/Remove Windows Components.

Once you have your system ready for the CSK, it's time to download the starter kit from <http://www.asp.net/starterkits>.

Before downloading the starter kit, you are offered the option to create a user account for the ASP.NET Forums. The CSK discussions provide useful insight about developing solutions using the CSK.

After you go to the Download Starter Kit link in the CSK section, you'll need to choose what version of the starter kit to download.

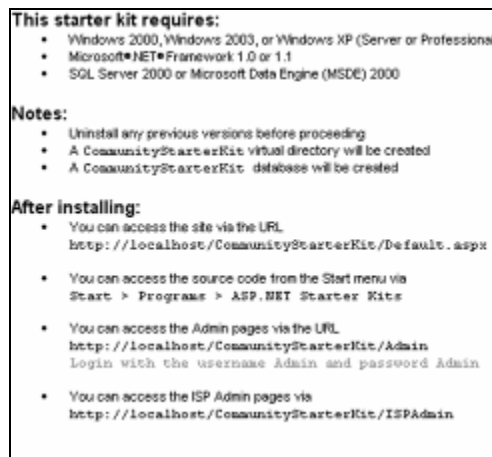
There are six versions of the CSK available for download for the three standard .NET languages provided by Microsoft: C#, VB.NET, and J#. For each of these languages, you can download the SDK version or the VS (Visual Studio .NET) version. The SDK version doesn't use code-behind files and is more appropriate for development environments like Web Matrix.

Web Matrix is a free ASP.NET development tool that you can download from <http://www.asp.net/webmatrix> and its installer kit is only 1.3 MB. It doesn't support all the features you can find in Visual Studio .NET, such as support for code-behind files, IntelliSense, or debugging, but can be very useful for developing small ASP.NET applications. For more information about Web Matrix, visit:
<http://aspnet.4guysfromrolla.com/articles/061803-1.aspx> or
<http://www.aspnetpro.com/productreviews>

There is no difference in functionality between the different versions of the starter kit. The only difference is the language they're written in and the way they are packaged.

This book focuses on the **CSVs** version of the starter kit, which is a C# ASP.NET web application for Visual Studio .NET. The file download for this starter kit is named ASP.NET Community (CSVs) Installer v1.0.msi.

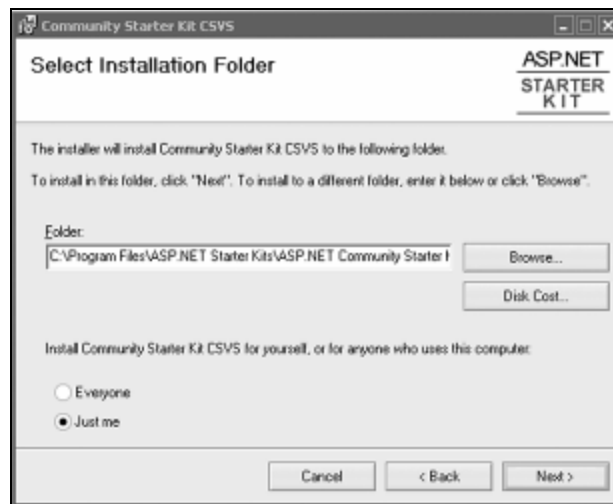
After downloading the MSI file, the installation shouldn't raise any problems. At the beginning, you're presented with some useful details about the system requirements and the installation location:



The actual web locations, admin name, and password may differ since you have the option to change them during installation.

After accepting the license agreement for Microsoft ASP.NET Starter Kit, you'll have the option to specify:

1. The physical location on the hard drive where the CSK files will be installed:



2. The web location where the starter kit will install. For this book, we suggest that you leave the default virtual directory name (Community Starter Kit) unchanged, making it accessible through `http://computername/CommunityStarterKit`. However, changing the web folder name won't make much of a difference.



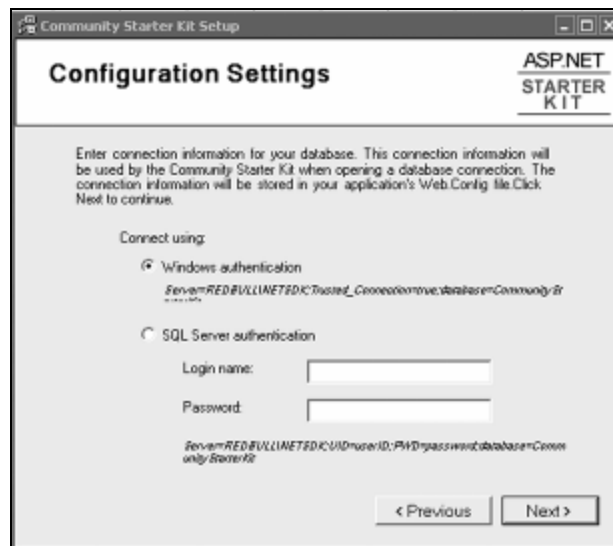
3. The MSDE or SQL Server login information that will be used by the installer to create the necessary database objects. Obviously, the user account you specify must have permissions to create databases or objects in a specific database (if you'll choose to use an existing database).



4. The name of the CSK database. You can choose to create a new database (which is the recommended option if you have the necessary privileges), or you can use an already existing database.



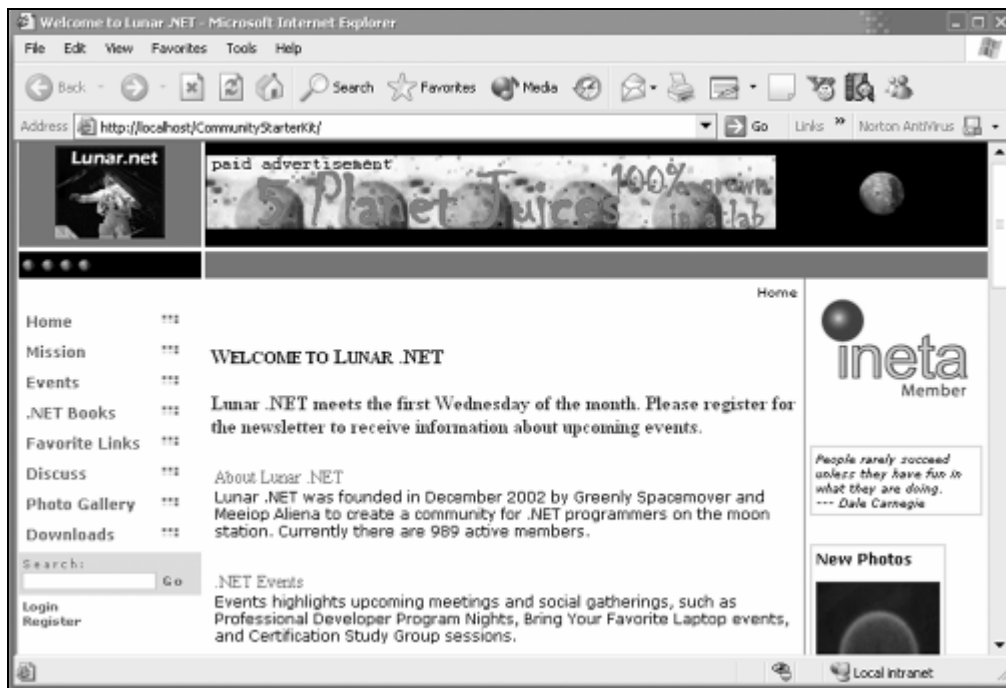
5. The login information that the CSK will use to access the CommunityStarterKit database. This doesn't have to be the same as what you supplied earlier when installing the database. Here you can supply a username with rights only to the CommunityStarterKit database.



6. ISP admin username and ISP admin password data (which, by default, are ISPAdmin and ISPPassword respectively), and community admin username and community admin password (by default, both are Admin).

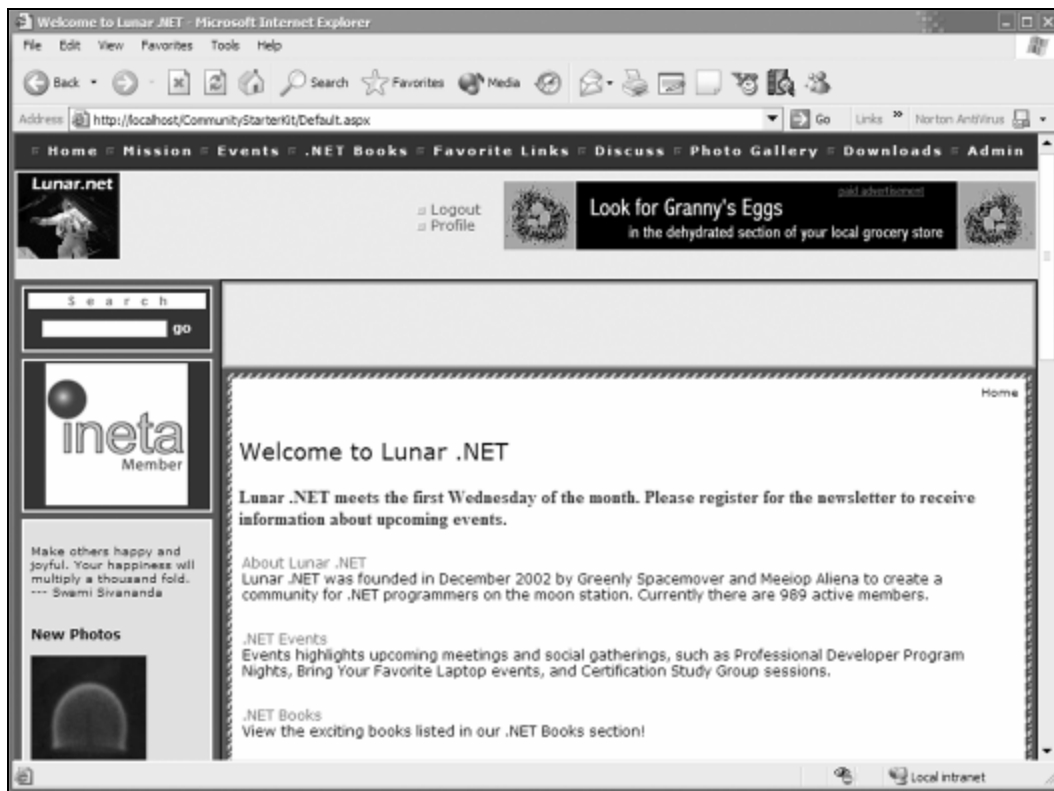
A single CSK installation can support multiple communities. During the setup, the installer creates a single sample community whose administrator account is Admin. Admin is a local community account and has privileges only inside its community. The ISPAdmin account, on the other hand, is not part of any community. ISPAdmin is used to access the ISP administration pages, where you can manage the communities in the CSK installation.

That's it—the CSK is now ready for action! The installer also populates the database with a sample community named Lunar .NET (an imaginary community for .NET programmers on the moon station) that you can see in action right away. If you haven't changed the default virtual directory name, you can visit the Lunar .NET community at <http://localhost/CommunityStarterKit:>



Generally, the installation process works well; however, there have been some reports on the ASP.NET forums with problems installing the kit. Typical problems are related to the connection string used to access the CommunityStarterKit database. The connection string is stored in the web.config file in the CSK project folder. Also, the CSK installer program creates a log file named install.log in the setup folder. This is also a good place to look for the root of your problem.

As mentioned earlier in the chapter, you can change the appearance of the website using the administrative interface. This is because the underlying architecture is based on skins. With only a few clicks, it's possible to completely change the face of the site. The following screenshot shows one of the themes included in the starter kit:



Summary

In this chapter, we took the first step into the world of CSK. We briefly discussed what the Community Started Kit can do. We discussed the system requirements necessary to install the CSK and went through the installation process.

This starter kit can prove to be a very useful tool in your web development toolbox. It was designed with both usability and reusability in mind, and with its powerful and flexible infrastructure, building customized community websites becomes a straightforward task. Our next step will be towards learning how to customize and extend the CSK.

In the next chapter, we'll take a journey through the features bundled with the CSK.

2

Touring a Community Starter Kit Website

It's time to make a trip to the moon. Touring the Lunar .NET community, the sample community website that ships with the Community Starter Kit (CSK), will help you learn about the features and functionality bundled in the starter kit. The first step towards creating your own community website based on the CSK is to get accustomed to its functionality.

Lunar .NET implements almost all the functionality supported by the starter kit engine. After this tour, you might find that the CSK includes everything you need for your own website and more.

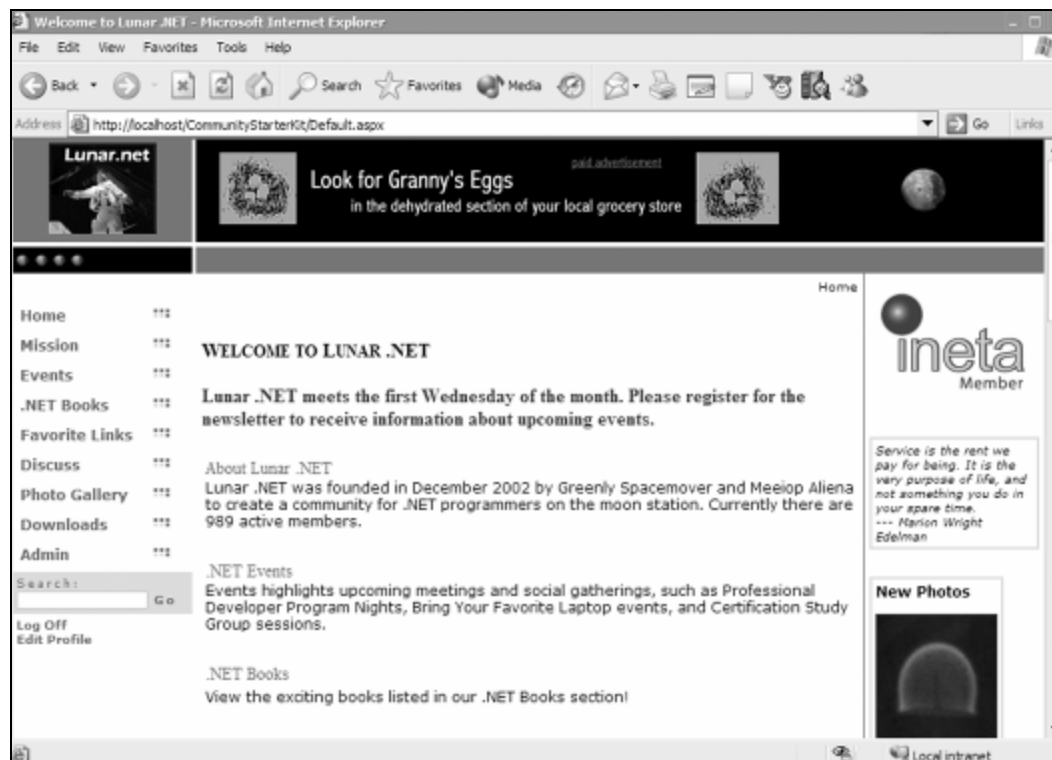
Chapters 3, 4, and 5 will teach you how to create a community website by reusing what you already have. In the later chapters, you'll learn how to extend the CSK by creating new modules.

In this chapter, we aim to:

- Analyze the general features provided by all major CSK modules
- Learn how to create and edit new user accounts
- Look at how each CSK module works

Overview of the Community Starter Kit

Let's load the Lunar .NET community in Internet Explorer. If you installed the CSK with the default options, you can access it at <http://localhost/CommunityStarterKit>. The following screenshot shows the default look of the main page of the Lunar .NET community:



It's assumed you're running the CSK on your local machine. If this is not the case, you'll need to refer to the machine that hosts the remote community instead of localhost. Also, if you chose to use a different virtual directory when installing the kit, you'll need to use that name instead of CommunityStarterKit in the URL.

The Main Page

The main page in the Lunar .NET community contains the elements expected of any community website: a logo, a menu with links to various sections, an advertising banner, and so on. In this chapter we learn how the built-in functionality works, and in Chapter 3 we'll learn how to administer it through the Community Administration Menu.

Bear in mind that it's possible to completely change the layout of a CSK website by building new skins (as we'll learn in Chapter 4) or using the predefined themes that ship with the CSK.

You can preview the predefined themes at
<http://localhost/CommunityStarterKit/Admin/help/theme.htm>.

The Community Logo

The community logo appears in the upper left part of the site. It can be changed by editing the logo of the home page in the Community Administration Menu.

The Community Menu

The community menu is just below the logo, and contains links to the sections of your community. Each section in the community has a **content type**, such as Books, Links, or Photo Galleries. Each content type is governed by a specific CSK module (such as Books module, Articles module, and so on).

You can have several sections in your community website with the same content type. For example, you can have two Books sections named "C# Books" and "VB.NET Books", both built using the Books module.

We'll discuss the CSK modules and the different types of content supported by the CSK later in this chapter.

The Search Box

The Search box permits searching for content in the section that is being visited or in all public sections of the website. The following screenshot shows this feature in action:



We'll discuss how the search functionality works internally in Chapter 6.

Login/Register

The main page features the Login and Register links for unauthenticated users and the Log Off and Edit Profile sections for logged in users. Their functionality is obvious, and we'll analyze them in detail in the *Creating and Using User Accounts* section later in this chapter.

Advertising

At the top and in the right of the main page, you can see two advertising boxes generated by AdRotator controls. In Chapter 3, we'll discuss how to configure the ads to be displayed.

Web Boxes

Web Boxes are used to display information outside of the main content area of your pages. In the Lunar .NET community site, by default, you can see a random quote box and a list of new photos in the right side of the page.

Web Services Boxes

Web Services Boxes are similar to Web Boxes, except that they get their source information from an external RSS feed. By default, you can see information gathered from the official Star Wars website in the right part of the page.

Voting Polls

The community administrator can add voting polls through the Community Administration Menu. You can see in the following screenshot how the voting poll looks by default:



The Contents

The main page content area changes depending on the type of community section being viewed. The home page, which is a parent section, shows links to the other sections of the website. If you go to one of these sections, the contents section will change accordingly. We'll see more about each kind of website section later in this chapter, and in Chapter 3, we'll see how to change these sections using the Community Administration Menu.

Overview of the CSK Modules

Now let's learn more about each type of content supported by the CSK by playing further with the Lunar .NET community.

In the following sections, we will look at:

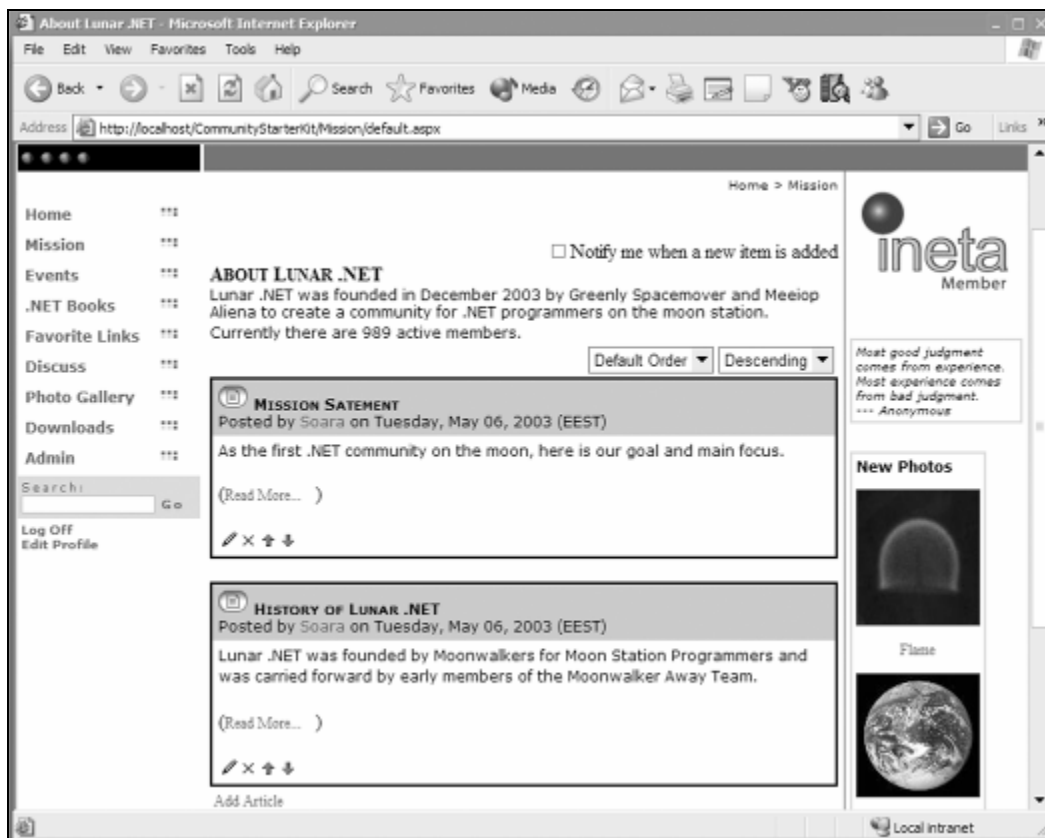
- Articles
- Events
- Books
- Links
- Discussion Forums
- Photo Gallery

- Downloads
- Parent Section
- HTML Page

In this chapter, we aim at having a good overview and knowledge about the capabilities of these modules. We will discuss their functionality with more details in Chapter 3 while learning how to administer them using the Community Administration Menu.

Articles

The Articles page of the Lunar .NET community can be accessed by clicking on the Mission link. The Articles module first shows a list of the articles in the current section:



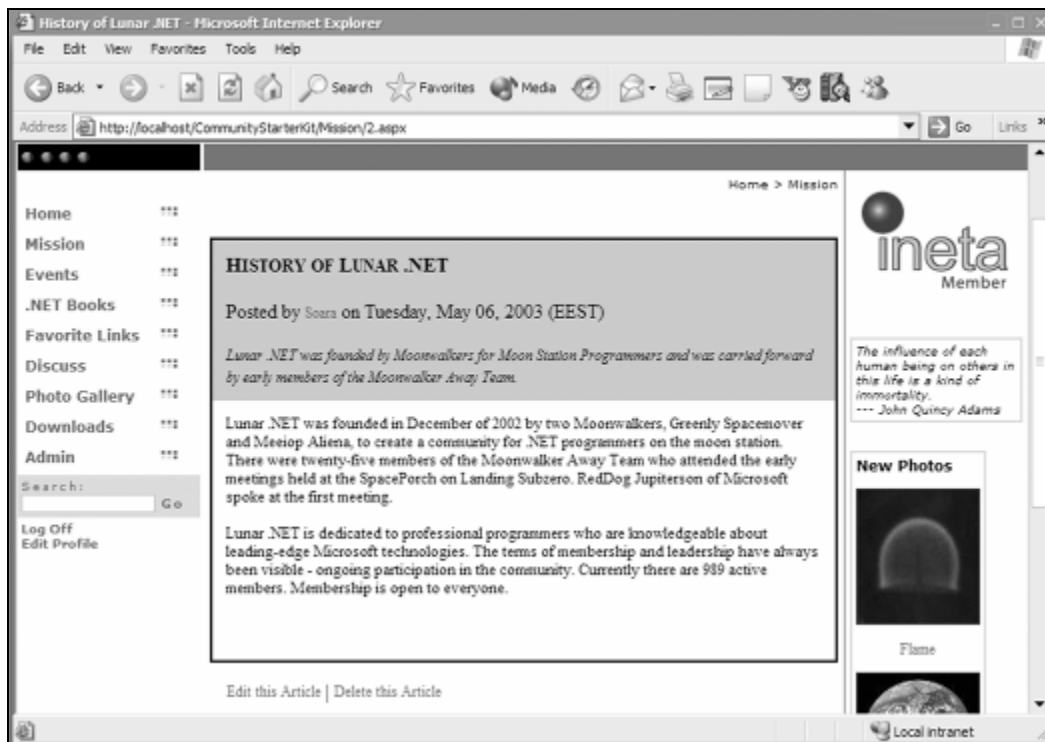
While logged in as Admin, apart from the usual content, you can also see the following elements:

- The Admin link, which leads to the Community Administration Menu

- The edit, delete, and move controls that appear for every article in the section
- The Add Article link, which allows adding a new article to the section

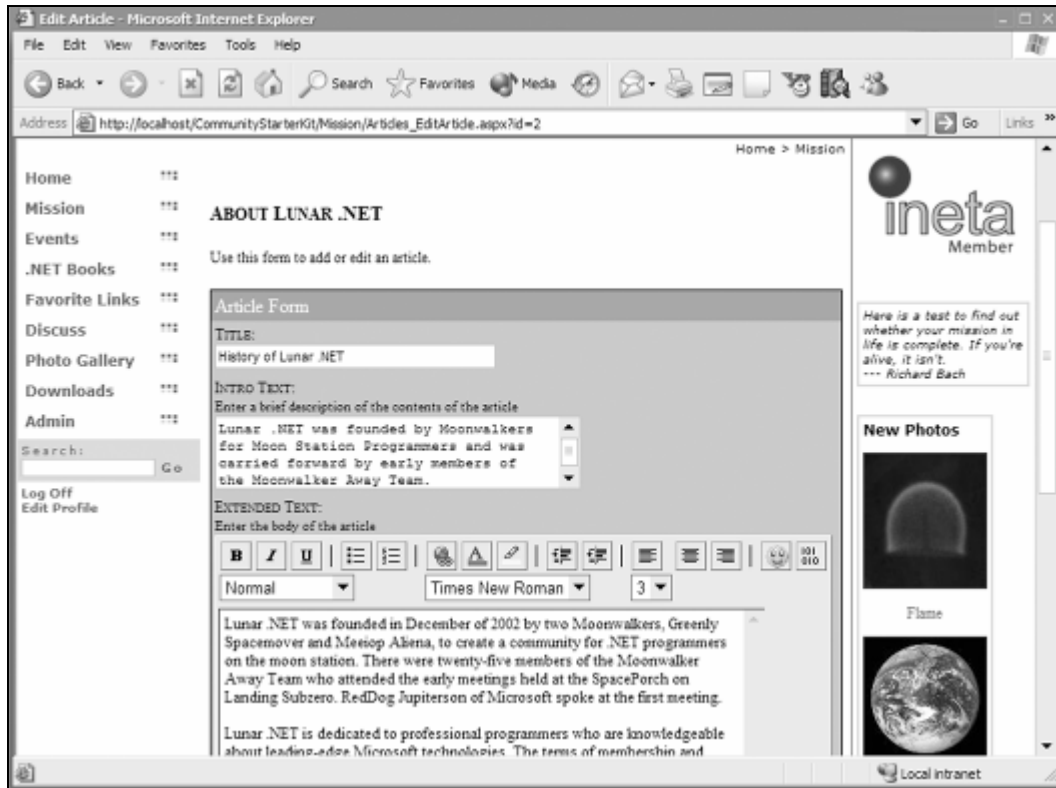
The roles that are allowed to read, edit, delete, or add articles for a section can be set by the administrator in the Community Administration Menu | Edit Sections | Edit Section | Security menu. However, the administrator account always has those privileges.

When the Read More link is clicked, the article details page is shown:



Note the Edit this Article and Delete this Article links, which appear only when you are logged in as administrator.

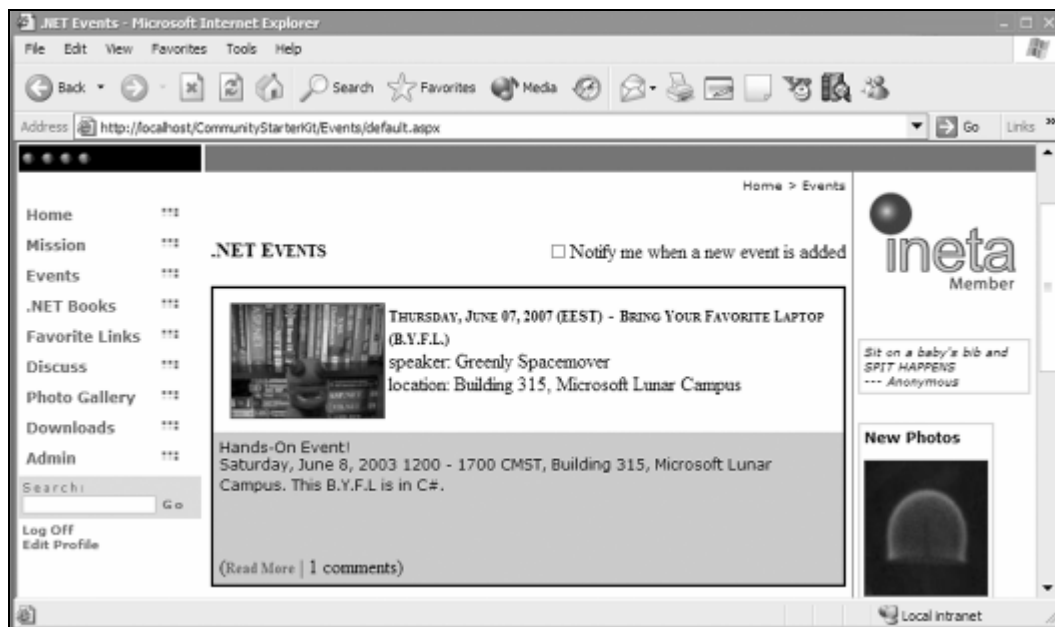
When editing or adding a new article, an easy-to-use interface is provided:



Feel free to play around a little bit and get used to adding and updating articles. Note that for writing in the Extended Text section, you are provided with the **HTML Editor** (we will look at this in detail later in the chapter).

Events

The Events section shows the upcoming events in your community. You can access this section of the site by clicking on the Events link. The following screenshot shows the display of an event in the Lunar .NET community:



The Events section works in a way similar to the Articles section, except:

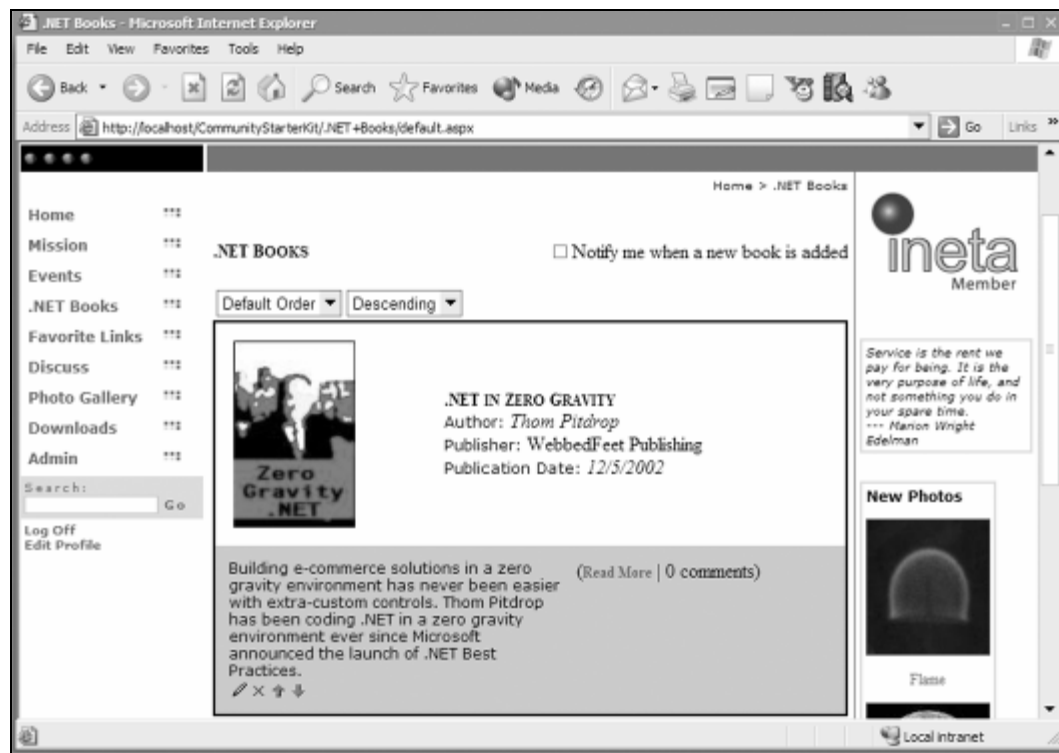
- You can associate an image with every event.
- You can schedule an event to appear only after a certain date.
- In order to edit or delete events, you need to go to the event's details page by clicking on its Read More link.

There is a View Past Events link that lets you see events that expired and are no longer shown in the main Events page.

Events are automatically ordered by date, so you aren't provided with up and down links to modify the order in which they appear.

Books

You can access the Books section of the site by clicking on the .NET Books link. The Books section is shown in the following screenshot:

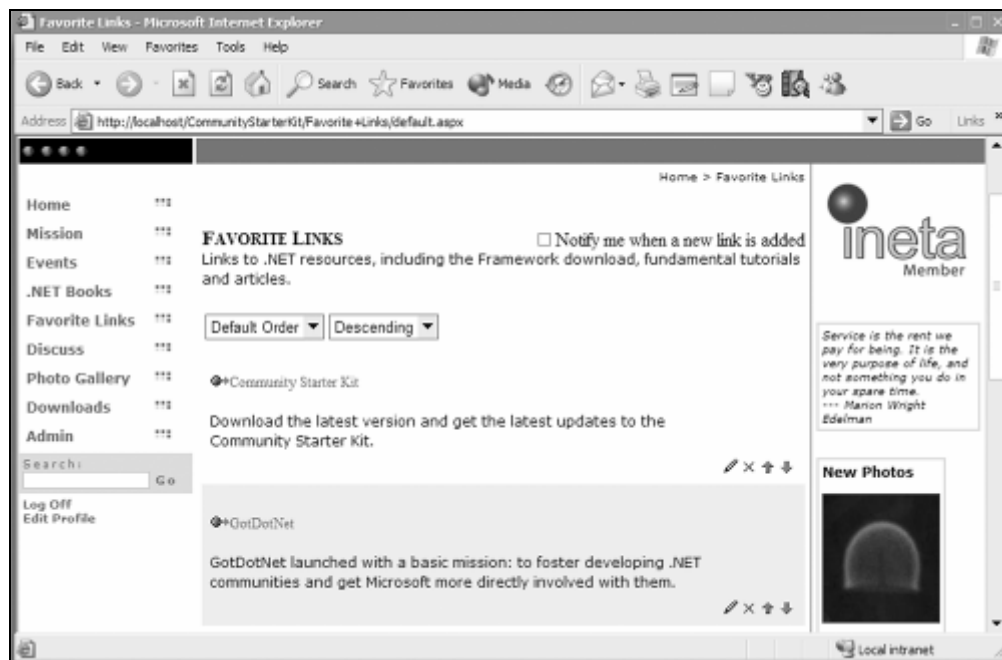


The Books module is similar in functionality to the Articles module, except:

- You can associate a cover image with each book.
- You can add a link to a place where the book can be bought.
- Each book includes further details such as its title, author, price, and so on.

Links

You can access the Links section of the sample website by clicking on the Favorite Links link in the menu. Each link object has a title, URL, and description. By default, there are five link objects listed per page. The administrative buttons allow you to edit, delete, or change the order of links:



Discussions

The Discussions module is used to host a discussion forum.

Discussions contain threads, and threads contain messages. Users can post new threads in a discussion or new messages in a thread.

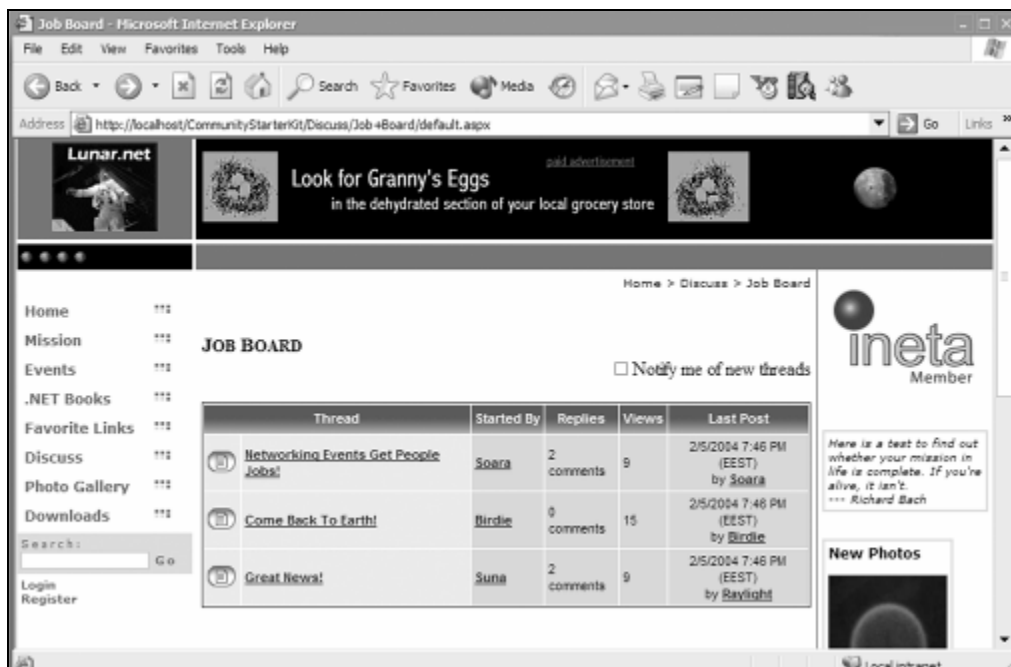
On clicking the Discuss link in the menu, you get the Discuss page that contains links to the three discussions (the Private discussion is only visible to community administrators.)

The Discuss page is a parent section that contains three separate discussion topics. Each of those discussions is governed by the Discussions module. The list of discussion topics can only be modified from the Community Administration Menu, which allows adding, removing, and changing website sections.

Touring a Community Starter Kit Website



The following screenshot shows the threads in the Job Board discussion:



Users that visit the discussions can:

- Post new threads. However, this option is affected by security options set for that section. In Lunar .NET, only administrators can post new threads in the Job Board forum, but everyone can post new threads in the Discussions forum.
- Reply to messages that exist in current threads.
- Post locked threads by selecting the Do not allow replies to this post checkbox when adding a new thread. When a thread is locked, other users aren't allowed to respond to that thread.
- Post announcements by selecting the Announcement checkbox when adding a new thread. An announcement is a thread that is marked with a special icon (an exclamation mark) when listed.
- Post pinned threads by selecting the Pinned post checkbox when adding a new thread. Pinned posts always appear at the top of a Discussion section.

Announcements and pinned posts are only allowed for users that have the privileges to edit or moderate the selected discussion.

The Discussions module is a simplified version of the AspNetForums, which is also available for download at <http://www.asp.net/>.

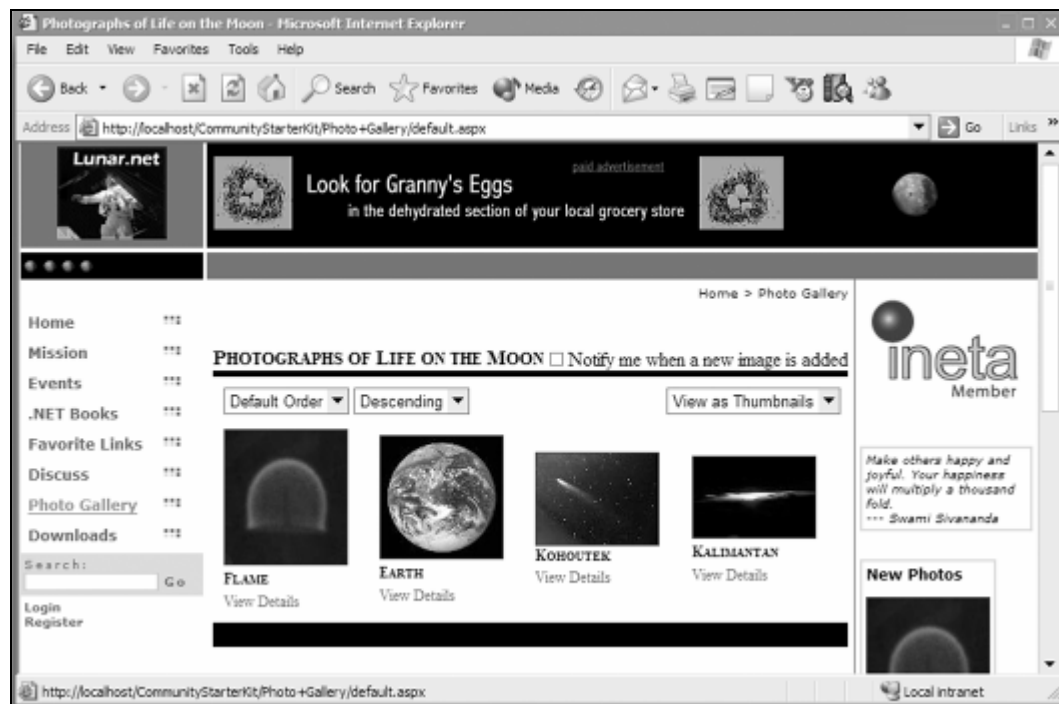
Photo Gallery

The Photo Gallery contains photos for your community. A Photo Gallery module can store a number of pictures for the community. When adding a new picture, a thumbnail is automatically generated and stored.

With the default View as Thumbnails layout, only the thumbnails are shown when opening the Photo Gallery, while the enlarged picture is displayed when a particular thumbnail is selected.

The Photo Gallery section enforces both user and community quotas (see the *Quotas* section later in this chapter). A new picture cannot be added if the user adding it, or the community as a whole, exceeds the quota allocated for them.

You can access it from the Photo Gallery link in the menu:



Each photo has the following characteristics:

- Image file
- Name
- Caption
- Full description

Downloads

The CSK stores files for users to download. Access the Downloads section in the Lunar .NET community by clicking the Downloads link in the menu.

The files in the Downloads section enforce both user and community quotas.

When adding a new download, you need to specify:

- The name that the file will be stored as on the community server, which can be different from the original file name. Remember that the files are stored in the database, not in the Windows file system.
- Brief description (that shows in the list of downloads).
- Full description (that appears when seeing the details of a particular download).
- Path to the file (the complete path to the original file).

The following screenshot shows the form for adding a new download:

The screenshot shows the 'Add File Form' on the Lunar.net website. The page has a dark header with the Lunar.net logo and a 'paid advertisement' banner. A sidebar on the left contains navigation links: Home, Mission, Events, .NET Books, Favorite Links, Discuss, Photo Gallery, Downloads, and Admin. Below these is a search bar and links for 'Log Off' and 'Edit Profile'. The main content area is titled 'Add File Form' and includes a 'Use this form to upload files to the portal.' instruction. It displays user and community quotas: 'User Quota: 10,000 KB' (Used: 2,086 KB) and 'Community Quota: 10,000 KB' (Used: 2,433 KB). The form fields are: 'FILE NAME' (with a hint '(For example, ex7File.exe)'), 'FILE BRIEF DESCRIPTION' (with a hint '(This information is listed on the Downloads page)'), and 'FILE FULL DESCRIPTION' (with a hint '(Users can access this information when they press the "Read More" link)'). Below the descriptions is a rich text editor with formatting tools (bold, italic, underline, list, link, unlink, image, table, quote, code, undo, redo) and a font dropdown set to 'Times New Roman'. At the bottom, there is a 'FILE' input field (with a hint '(For example, C:\File\myFile.exe)'), a 'Browse...' button, and an 'Add File' button. On the right side, there is a 'Home > Downloads' breadcrumb, an 'ineta Member' profile, a quote 'May the smile on your face Come straight from your heart --- Anonymous', and a 'New Photos' section with three image thumbnails labeled 'Flare', 'Earth', and 'Kekeutsk'. A footer at the bottom states: 'NASA is acknowledged as the source of the photographs of space. No endorsement by NASA is implied.'

The maximum length of the file being uploaded is also limited by the value of the `maxRequestLength` attribute of the `httpRuntime` element in the `Web.Config` file (the configuration file located in the CSK installation folder). This value specifies the maximum upload size in kilobytes, and the default value in CSK is 10000 (overriding the ASP.NET default of 4000). You may want to add the `executionTimeout` attribute to extend the maximum time a request is allowed to execute (the default is 90 seconds). We'll discuss the `Web.Config` file in detail in Chapter 6.

Parent Section

Parent sections are used to include other sections named **child sections**. In the Lunar .NET community, you can find two parent sections, the Home page and the Discuss page. The Parent

Section page contains links to, along with a short description of, its child sections. Each section except Home must have a parent section.

HTML Page

The HTML Page module permits you to post a customized static HTML page to your website. After adding the HTML page to your website, you have the option to edit it using the built-in HTML Editor.

Let's continue by looking at some important features that are common to many of these modules.

Common Features in the CSK Modules

The following list summarizes the most important CSK features:

- Security
- Moderation
- Comments
- Ratings
- Notifications
- Quotas
- HTML Editor

Before we can talk about each community module, it's important to know what these features do, even if you will not understand all the details now. While analyzing the individual community modules (like Articles, Photo Gallery, and so on) later in this chapter, we'll make frequent references to these features.

Security

Security controls "who-can-do-what" in a particular environment. Security in general, and with the CSK in particular, deals with two fundamental notions:

- **Authentication:** Deals with uniquely identifying users by (usually) asking for a username and password
- **Authorization:** Decides what permissions an authenticated user has while browsing the community website

The CSK implements authentication by adding support for **user accounts**, so it can authenticate them when necessary by asking for a username/password combination. The CSK implements authorization through the notion of **roles**. A role is a named set of permissions.

Roles contain permissions, and usernames are attached to roles.

Roles are useful because they ease the administration of security inside the community. Instead of setting the individual permissions for each user, you assign permissions to roles and then assign roles to users. Thus, when changing the security permissions of a particular role, all the users that belong to that role are automatically affected.

Let's see what this theory means in practice for you and for the CSK.

Every community created with the CSK by default contains four **system roles** described below. A system role cannot be removed from a community.

- **Community-Administrators** is a role for the administrators of the community. They have access to the Community Administration Menu (we'll describe this in the next chapter), and they are allowed access to additional controls while browsing the community website. This allows them to change the section contents.
- **Community-Moderators** is a role assigned to content moderators. In the CSK, you can set many sections—the Articles or Books sections for example—to accept feedback (messages) from users. If a section is moderated, the content added by users (such as messages, articles, photos, downloads, and so on) will not show on the site without a moderator's approval.
- **Community-Everyone** is a special role that is automatically assigned to all visitors of your community, even if they don't have a community account. When assigning this role to a particular feature, every visitor will be able to access that feature on your website.
- **Community-Authenticated** refers to users that have an account in your community and require to be authenticated.

For example, you're likely to allow only authenticated users (by specifying the Community-Authenticated role) to post new articles on your website or post messages in these articles. However, by specifying the Community-Everyone role, you're free to let everyone perform these actions.

Apart from the four system roles, you can define other roles for your own particular needs. For example, you can create roles like Books-Moderators and Articles-Moderators and assign these roles to moderate new contents in the Books and Articles sections. We'll learn how to do that in Chapter 3, but for now it's good to bear in mind that this possibility exists.

Note that a community administrator can assign other users to the Community-Administrators or Community-Moderators roles, or to any other user-defined roles. However, Community-Everyone and Community-Authenticated roles are automatically assigned to users by the CSK system.

Moderation

If moderation is enabled for a section, a moderator must approve pending content posted by other users before that content is displayed.

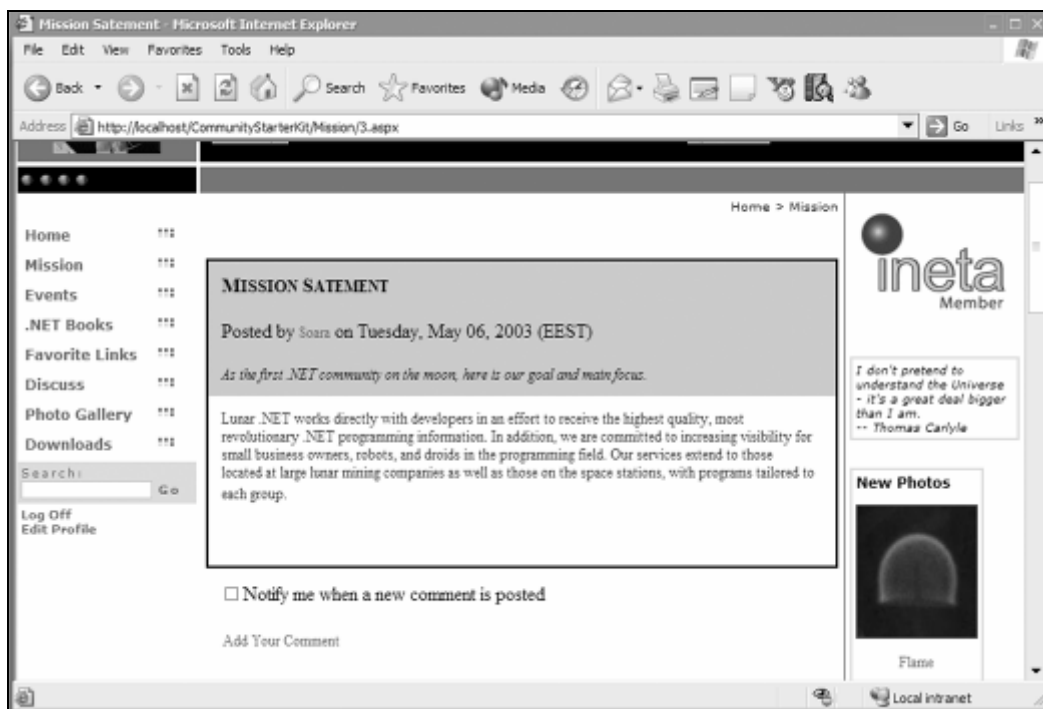
Moderation rules don't apply to moderators, whose content is displayed immediately without waiting for approval.

Let's review the most important aspects about moderation:

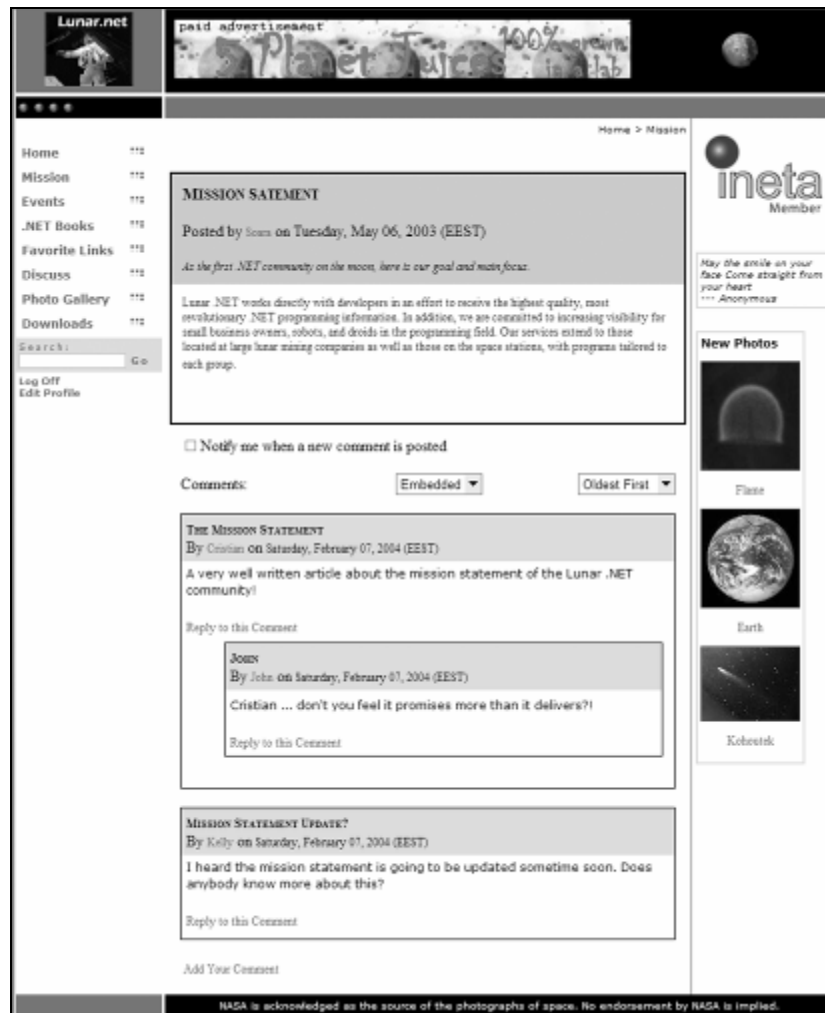
- The CSK contains a system role named Community-Moderators that is meant to be used for moderation purposes.
- Users in the Community-Administrators role can create other roles and allow them to moderate one or more sections of your community.

Comments

You can enable comments for various sections of your website (such as Articles, Books, and so on). When comments are enabled, the users that are part of a role that is allowed to add comments will see an Add Your Comment link when viewing articles from that particular section:



A comment, after it's added, appears below the article message. When adding a new comment, you can either reply to existing comments or post new ones:



When viewing comments, the visitor can choose from four display modes: Flat, Nested, Threaded, and Embedded. The preceding screenshot shows the Embedded mode, where a comment to another comment is displayed *inside* the parent comment.

The CSK does not provide the ability to delete or edit comments.

Ratings

When ratings are enabled for a module in the website, users in the selected roles can rate the content of the section with marks from 1 (Bad) to 5 (Good). Comments that were posted in those sections can also be rated.

The following screenshot shows the article rating controls. Before any votes exist for a section, the default rating is 3:



Once a rating has been made, you can see both your rating and the average rating:



Notifications

When notifications are enabled for a section in the CSK website, authenticated users can choose to receive e-mail notifications when new content is added to that section.

The following screenshot shows the Home | Mission section, with the checkbox that allows the user to enable e-mail notifications for this section:



The checkbox also appears when visiting individual articles in a section.

When creating a new account, the user can choose not to receive e-mail notifications. In this case, the user will not see the Notify me when a new item is added link even for the sections that have notifications enabled.

Quotas

Each user of a community has a fixed quota for adding images or downloads to the community and cannot exceed that quota. The community administrator can set the individual quota for a particular user from Community Administration Menu | Edit Users.

The quota for the whole community can be set by the ISP administrator. We'll see in Chapter 3 how to set the quota for a community.

The CSK doesn't store images and downloads in the file system, but in its SQL Server database. The quotas are enforced at the database level.

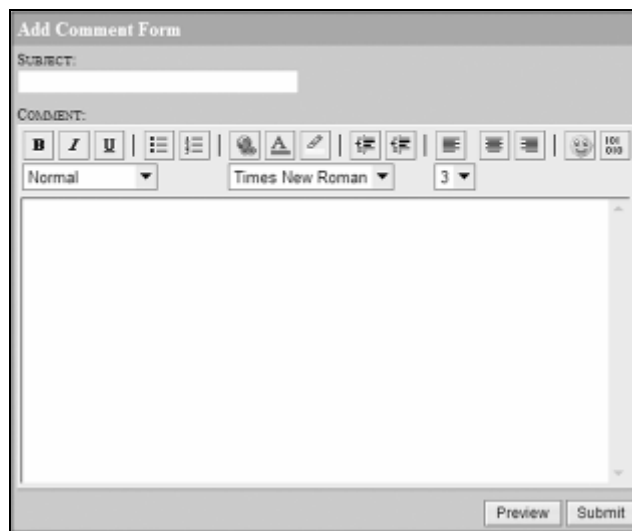
We will talk about the infrastructure that makes the CSK work in Chapter 6.

HTML Editor

The CSK provides an HTML Editor that can be used when adding or editing content (such as articles, comments to articles, or book descriptions), or when adding or editing comments.

The HTML Editor works as advertised only with Internet Explorer 5.5 or newer. For older versions of IE or for other browsers (such as Opera and Mozilla among others), the HTML editor will show a simple textbox without HTML support.

In the following screenshot, you can see the form that appears when you can enter HTML content. Note the buttons in the control's toolbar provide the functionality of a simple text editor.



The HTML Editor is just one of the many controls that ship with the CSK. We mentioned it here because of its complexity and usefulness. We'll learn more about the CSK controls in Chapter 4.

The administrator can set the level of HTML support for each section in the website from Community Administration Menu | Edit Sections | Edit Section | Advanced. The three levels of HTML support are:

- **Full:** HTML Editor provides an additional View HTML Source checkbox that allows manual editing of the generated HTML.
- **Limited:** With limited support, visitors can add HTML to their messages only using the toolbar buttons in the HTML Editor. They aren't allowed to edit the generated HTML (which could be a problem, because they would be able to insert JavaScript code, and so on). Limited is the default mode of operation of the HTML Editor.
- **None:** With HTML support disabled, the control will show only a simple textbox.

Creating and Using User Accounts

Because some features in the CSK modules can be accessed by default only by authenticated users, you need to know how to create a new user account. You can also cheat a little bit by using the administrator account; with this, you can unlock features that let you see how to change or add content directly in the community web pages.

Keep in mind that adding or editing content (for example, modifying articles, book information, or links) doesn't imply accessing the Community Administration Menu.

Creating a New User Account

In the left part of the community website, you can see the Register link. After clicking on it, you're presented with a form where you can enter the required data for your new account:

Lunar.net paid advertisement Planet Juices 100% organic in a lab

Home

Home Mission Events .NET Books Favorite Links Discuss Photo Gallery Downloads Search: Go Login Register

REGISTER

Complete the following form to register.

Required Information

USERNAME:

PASSWORD:

PASSWORD AGAIN:

EMAIL ADDRESS:

FIRST NAME:

LAST NAME:

TIMEZONE:

Contact Information

MSN:

YAHOO:

AIM:

ICQ:

WEBSITE:

PUBLIC EMAIL:

Optional Information

OCCUPATION:

LOCATION:

INTERESTS:

Email Options

NEWSLETTER: ☒ Receive Newsletter

EMAIL NOTIFICATIONS: ☒ Receive Email Notifications
(This option controls Global Email Notification. If this option is enabled, you will see Email Notification checkboxes in each section of this community. Select checkboxes in desired sections to receive an Email Notification when information is added.)

New Photos

Flare

Earth

Kobresnik

NASA is acknowledged as the source of the photographs of space. No endorsement by NASA is implied.

Note that in the registration page, you have the option to choose receiving newsletters or e-mail notifications. Sending newsletters is the privilege of community administrators. After clicking the Register button, you are forwarded to the main community web page and logged in with your new user account.

Log In

If you already have a user account, you log in by clicking on the Login link on the left side of the screen. Since you have just created a new user, you will need to log off first.

If you don't do anything on the website for five minutes, you'll automatically be logged out.

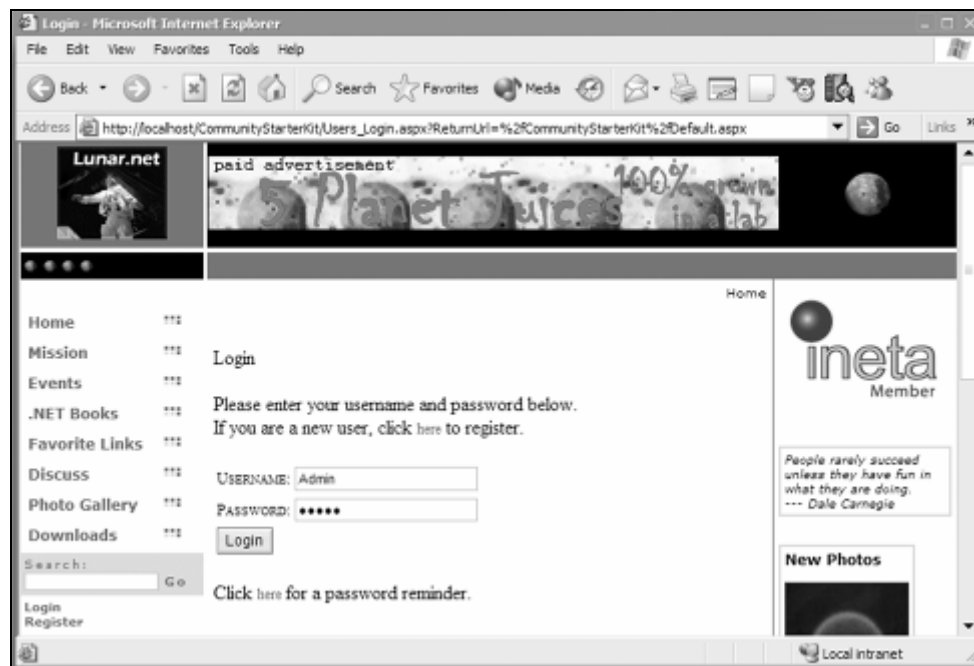
Also, when trying to visit a secured area in the website that requires any user role other than Community-Everyone, the Login page shows up automatically. In that case, after logging in, the user is automatically redirected to the initially requested page. Bear in mind that you should first log off if you want to access a secured area that the current user doesn't have access to.

Edit Profile

While logged into your user account, you'll see the Edit Profile link below the search box and the Log Off button. After clicking on Edit Profile, you are shown a summary of your account information. On that page, you have the option to click on a second Edit Profile link, which will actually allow you to edit your profile and lead you to a page similar to the registration page that we saw earlier; from this page you can modify your account preferences including your username.

Logging in as Administrator

If you haven't changed the login information when installing the CSK, the default administrator username is Admin and its password is Admin:



When logged in as Admin, you will see extra controls on the pages that allow you to change the site content. Also, in the menu on the left, an additional link named Admin will appear which redirects to the Community Administration Menu.

If you don't show any activity on the website for five minutes, you'll be automatically logged out. If you logged in as Admin and the special administrative controls or the Admin link disappears, it is a sign that you should log in again.

Summary

In this chapter, we had a high-level overview of the features of the CSK. We took a quick look at the CSK modules and the common features found in them. We also saw how to create a new user account, because some features in the CSK modules can by default be accessed only by authenticated users.

In the next chapter, we'll learn how to create new communities and how to create and manage the sections inside them (such as adding more Articles sections, and so on). This will be a big step towards learning how to create your own unique community website.

3

Creating and Administering Community Websites

At first glance, a web page consists of colors, layout, and data. When dynamically building the requested page for a visitor, the CSK engine imports these elements from external sources. You can easily change (or create) what the visitor sees by altering these external sources and without recompiling the project. The CSK stores the colors (and styles) in external Cascading Style Sheets, and the layout is managed by skin files (you'll learn about these in Chapter 4.)

This chapter will teach you how to administer the data displayed by the CSK. We will specifically talk about:

- Using the Community Administration Menu to manage your community
- Managing global community data and working with multiple communities on a single CSK installation by using the ISP Admin pages

Using the Community Administration Menu

The CSK is equipped with a powerful administrative interface named the Community Administration Menu. It is an excellent tool that allows you to control and change your community without changing a single line of code. It is organized into multiple administrative sections that control various aspects of the community website. However, the Community Administration Menu has a few weaknesses:

- You cannot add or remove new users through the Community Administration Menu. The administrator can edit existing users through the Community Administration Menu | Edit Users section. A new user can register through the Register link in the main page (we discussed this in Chapter 2).
- You cannot edit the list of quotes.
- You cannot administer the advertising banners.

These limitations can be overcome in four main ways:

- Manually editing external configuration files (XML configuration files, `web.Config`).

- Editing or adding skin files. Skin files permit changing the layout (and sometimes, the data) that appears in a community page without modifying a single line of CSK code. You'll learn more about skins in the next chapter.
- Manually updating the CSK code or the CSK database. You'll start learning how to do this in Chapter 6.
- Adding custom CSK modules or functionality. You'll learn how to do this in Chapters 7 through 10.

To access the Community Administration Menu, you need to be logged in as an administrator. Click on the Admin link that appears on the menu or type the URL `http://localhost/CommunityStarterKit/Admin` into the address bar.

If you go to `http://localhost/CommunityStarterKit/Admin` without being logged in as the administrator, you'll be redirected to the Login page. However, if you're logged in as another user, you must log out before trying to access the Admin page. The Community Administration Menu is shown in the following screenshot:



The Community Administration Menu contains the following sections:

- Edit Sections
- Edit Voting Polls
- Edit Service Subscriptions
- Edit Messages
- Edit Topics
- Edit Users
- Edit Web Box Services
- Send Newsletter
- Edit Images
- View Activity Log
- Delete Content
- View Documentation

Let's look at each of these sections in detail.

Edit Sections

Edit Sections is perhaps the most important and complex part of the Community Administration Menu. It allows you to edit the list of sections in your community website, and their properties.

This page displays a list containing the current sections of the website. Although not explicitly shown in the list, each section has one of the content types that we discussed in Chapter 2.

Home	Edit	Delete	Sort Up	Sort Down	Section Name	Section Path
		X			Home	/Community/Start/1/default.aspx
		X			About	/Community/Start/1/About/default.aspx
		X			Books	/Community/Start/1/Books/default.aspx
		X			NET Books	/Community/Start/1/NET+Books/default.aspx
		X			Favorite Links	/Community/Start/1/Favorite+Links/default.aspx
		X			Discuss	/Community/Start/1/Discuss/default.aspx
		X			Discussion	/Community/Start/1/Discussion/default.aspx
		X			Job Board	/Community/Start/1/Discussion+Board/default.aspx
		X			Private	/Community/Start/1/Discussion/Private/default.aspx
		X			Photo Gallery	/Community/Start/1/Photo+Gallery/default.aspx
		X			Downloads	/Community/Start/1/Downloads/default.aspx

Add New Section | Return to Admin | Return to Site

Let's take a look at some of the functionality offered by the Edit Sections page:

- Clicking in the Home column of a section allows you to change that section to the home section. We will talk about this shortly.
- Clicking the Edit column permits you to edit almost any aspect of a section.
- Clicking the Delete column allows you to remove a section from the website.

- The up/down arrows let you arrange the sections in the list, which affects their appearance when listed in their parent sections. No matter how you order the sections, the home section will always be listed first.
- The Add New Section option lets you add a new section to the website. We'll see more details about this after analyzing the Edit Section pages.

Home and Discuss are parent sections and cannot be deleted without first deleting their child sections. Discuss is the parent section for the Job Board, Discussion, and Private sections. The other child sections are all children of the Home section.

Sections that are accessible to all visitors, without requiring special privileges, can be set as Home by clicking on the home symbol on the list:

- Any section that is made the Home section will be moved to the top of the list and it will cease to have a parent section.
- Child sections of the original home will now be children of the new home section.
- The original home section automatically becomes a child of the new home section.

Editing a Section

When you click on the Edit button for one of the sections, you enter the Edit Section page. It has six subsections which we'll look at one-by-one.

General

Here you can set general information about the section. This is depicted in the following screenshot:

The screenshot shows a web form titled 'Edit Section' with a vertical scrollbar on the right. The form contains several sections, each with a title, a description, and a text input field. The sections are: 'Section Name' (description: 'The name of the section is used to create the section path.'), 'Section Title' (description: 'The title of the section is displayed in the browser title bar and the body of the page.'), 'Section Menu Title' (description: 'The section menu title is displayed in menus.'), 'Section Description' (description: 'The section description is displayed when the section is listed in a parent page.'), 'Parent Section' (description: 'Select a parent section for this section.'), and 'Section Enabled' (description: 'Uncheck to hide this section.'). The 'Section Name' field contains 'Events', 'Section Title' contains '.NET Events', 'Section Menu Title' contains 'Events', 'Section Description' contains 'Events highlights upcoming meetings and social gatherings, such as Professional Developer Program Nights, Bring Your Favorite Laptop', 'Parent Section' is set to 'Home', and 'Section Enabled' is checked. An 'Apply Changes' button is at the bottom.

Section Name	Section Title	Section Menu Title	Section Description	Parent Section	Section Enabled
Events	.NET Events	Events	Events highlights upcoming meetings and social gatherings, such as Professional Developer Program Nights, Bring Your Favorite Laptop	Home	☒

Section Name is used in the creation of the section path that looks like the following:

`http://machine_name/CommunityStarterKit/section_name/default.aspx`

For example, you can access the section named Events through:

`http://localhost/CommunityStarterKit/Events/default.aspx`

Section Title shows up in the browser title bar and as the title of the section page. In the following screenshot you can also see the section menu title, which is the name that appears in the menu (and can be different from the Section Name):



Section Description appears by default on the section's parent page. It features the descriptions of all its child sections.

Parent Section lets you choose the parent of the section being edited.

Section Enabled allows you to temporarily disable a section by unchecking the checkbox. When a section is disabled it doesn't appear in the main page, in the menu, and it cannot be accessed by going directly to its URL (you get an HTTP 404 error). The CSK throws an exception when trying to use the content of a disabled section as the source for a Web Box, as explained in the article at <http://www.reddnet.net/Programming/CSK+Stuff/CSK+Files/CSK+1.0/337.aspx>.

Remember to click the Apply Changes button to save changes you make while editing a section.

Appearance

This section allows you to set various options that control the way a section is displayed.

Section Theme is the first and perhaps, the most interesting feature here. It allows you to set the **theme** by choosing the skin and style. You must choose these for the Home section, and for the subsections, you can choose either to inherit the skin and style from the parent section or select new ones. The skin establishes the layout of the page, and the style is a CSS file that sets font and color settings. We'll talk more about skins and styles in Chapter 4.

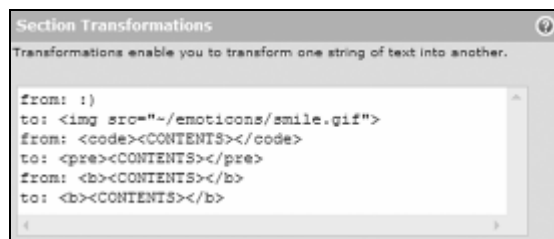
Section Logo is an image that shows up on your community pages. GIF and JPG (JPEG) are the supported file types (PNG is not supported). If you don't set up a section logo, it'll be inherited from the parent section. By default, the Lunar .NET community has a single logo image defined for the Home section. In order to remove an image that was added as the section logo, you need to access Community Administration Menu | Edit Images.

Section Page Header and Section Page Footer allow you to edit the text to be used as the header or footer of a section (using the HTML Editor that we saw in Chapter 2). By checking the View HTML Source checkbox, you can directly edit the HTML code.

Section Footer contains a line of text that appears at the bottom of the page. You cannot edit its HTML output unlike the Section Page Header or Section Page Footer. If you don't edit the Section Footer for a subsection, it'll be inherited from its parent section.

Section Meta Description and Section Meta Keywords are used by search engines to index your page.

Section Transformations defines rules that automatically transform one string of text into another when generating client output. For example, in any section content you'll see :) replaced by an emoticon. In the Lunar .NET community, the Home section defines a number of transformations (see the following screenshot), and these are inherited by all the other sections:



Web Boxes

Web Boxes are used to display information outside of the main content area of your pages (the default skin places it on the right side of the page). The information source can be:

- **A section from your own community:** In Lunar .NET, you can see a list of new photos that are provided by the Photo Gallery module.
- **A section from another community:** The CSK implements a system for exchanging information between communities using web services. You'll learn more about web services later in this chapter.
- **Really Simple Syndication (RSS) sources:** The CSK, by default, can only **import** from RSS feeds, but in Chapter 10 we'll learn how to extend the site to **expose** content through RSS (by default, the CSK only exports data through web services).

In the Lunar .NET community site, in the Web Box space, by default you can see a quote, a list of new photos, and information from the official Star Wars page.

Keep in mind that you're still in the Edit Section universe; this means you can choose different Web Box settings for each section in your community. For all sections except Home, you have the option to inherit the settings from the parent section.

The following screenshot shows the Web Boxes page in the administration menu that lets you choose these sources:



For subsections, you will see an additional option named Inherit from parent section, which is selected by default.

You can add more RSS sources from Community Administration Menu | Edit Web Box Services, which we'll discuss later.

There are some issues with Web Box behavior due to output caching. Please see <http://www.asp.net/Forums/ShowPost.aspx?tabindex=1&PostID=241330> for more details.

Security

The Security menu allows you to select the users who are allowed to view, add, edit, or delete pages for the section being edited.

For most sections in the Lunar .NET community, only the Community-Administrators role is allowed to add, edit, or delete content, while the Community-Everyone role is allowed to view the section pages. The Private discussion section is more restrictive and only Community-Administrators can view it.

The Security page lets you set add, edit, delete, or view permissions for a section as a whole. You can set permissions for moderating or adding comments or ratings for the section on the Advanced page.

Web Services

The CSK implements a communication mechanism for exchanging data with other community websites through web services. Your community can either expose one of its sections as a web service, or it can consume web services information from other community websites. Let's review what you know about web services and boxes.

First, don't confuse a web service with a Web Box service because they are quite different. Through web services you can populate entire sections in your community website from remote CSK websites (the remote content is automatically duplicated on the local community at regular intervals). Web Box services, on the other hand, populate the Web Boxes in your community with summary information and links from remote CSK websites or from an RSS feed.

The second important detail is that the CSK can expose data only through community web services. This data can be consumed by remote communities to populate their Web Boxes or to replicate your data inside their sections. However, the Photo Gallery and Downloads sections cannot be used with web services.

We'll see more about both kinds of services later in this chapter, in the Edit Service Subscriptions and Edit Web Box Services sections of the Community Administration Menu, where you can edit the list of remote sources. The Web Services page shown in the following screenshot allows you to choose if and how you want to use web services with the current section:



Remember that the information you see in the Web Services page is related to the community section you're editing. Most of the times you probably won't expose your data using web services at all; but if you want to use web services, you'll enable them only for the specific sections you want to make publicly available (note that you can set a password if you want to limit the external community websites that can access your web service).

To expose the section's content as a web service, check the Publish section content as a Web service checkbox.

To import remote content into the current section (the section for which you are editing the web services), you need to check elements from the All Items list. This list is empty by default, and it lists all remote web services that you set in the Community Administration Menu | Edit Service Subscriptions page.

Simply put, to import content into your community from another community, follow these steps:

7. Go to Community Administration Menu | Edit Service Subscriptions. There you can register the remote service, which may be something like `http://remote_machine/CommunityStarterKit/section_name/default.aspx`.
8. Create a new section in your website with the same kind of content as the section you want to import. For example, if you want to import events from another community, you need to create an Events section in your website.

9. Use the Web Services tab to import data from the registered remote service into the newly created section.

We'll see an example using this process later when we talk about the Edit Service Subscriptions page.

Advanced

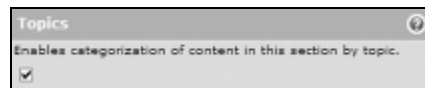
The Advanced tab contains a variety of options that you can set for the community section.

HTML Content establishes the work mode for the HTML Editor control when using it to edit or add content in the section being edited. You can set different options for content or content comments. The three HTML support modes are None, Limited, and Full; you learned about the HTML Editor control in Chapter 2.

Records Per Page allows you restrict the number of records to display per page when viewing the section. The default is 10:

The Comments dialog box lets you specify whether you want to enable comments for the section and the roles that are allowed to post comments. Usually, the Community-Authenticated role is allowed to post comments when comments are enabled:

By its nature, the CSK categorizes content by the content type (Articles, Books, and so on). The Topics box lets you enable categorization of content across multiple sections by topic even if the sections are of different types. This way, members of your community will be able to browse content by topic.



When topics are enabled for a section, users must specify a topic whenever they add content to that section.

The list of topics can be set from Community Administration Menu | Edit Topics (we'll discuss this page later). From here, you can also enable or disable the topic menu for the community. The topic menu contains links to the active topics in a community and allows easy browsing for community content based on topic. Topics that have just been declared but aren't used don't appear in the topic menu.

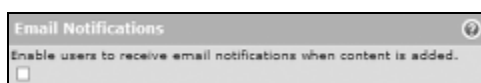
By default, the Lunar .NET community contains two topics: Politics and Internet. However, these topics are not used by default in any content. In the following screenshot, you can see the topic menu and two articles that belong to the Politics and Internet topics:



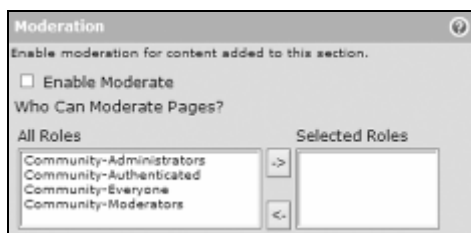
Here are a few things to keep in mind about the topic menu:

- It is refreshed every ten minutes. If you use a new topic when adding content, you'll need to wait ten minutes before it appears in the topic menu.
- It contains the topics that were used in the most recent 50 content items added to the database.
- Topics that were used more recently are listed first in the topic menu.
- When a topic link is clicked, a page that contains all content belonging to that topic is displayed.
- The list of available topics can be edited at Community Administration Menu | Edit Topics.

Email Notifications contains a checkbox that enables or disables e-mail notifications for the section. When this option is enabled, registered users who chose to receive e-mail notifications in their profile will see the Notify me when a new item is added checkbox when opening a content section. The notification message itself can be edited from Community Administration Menu | Edit Messages.

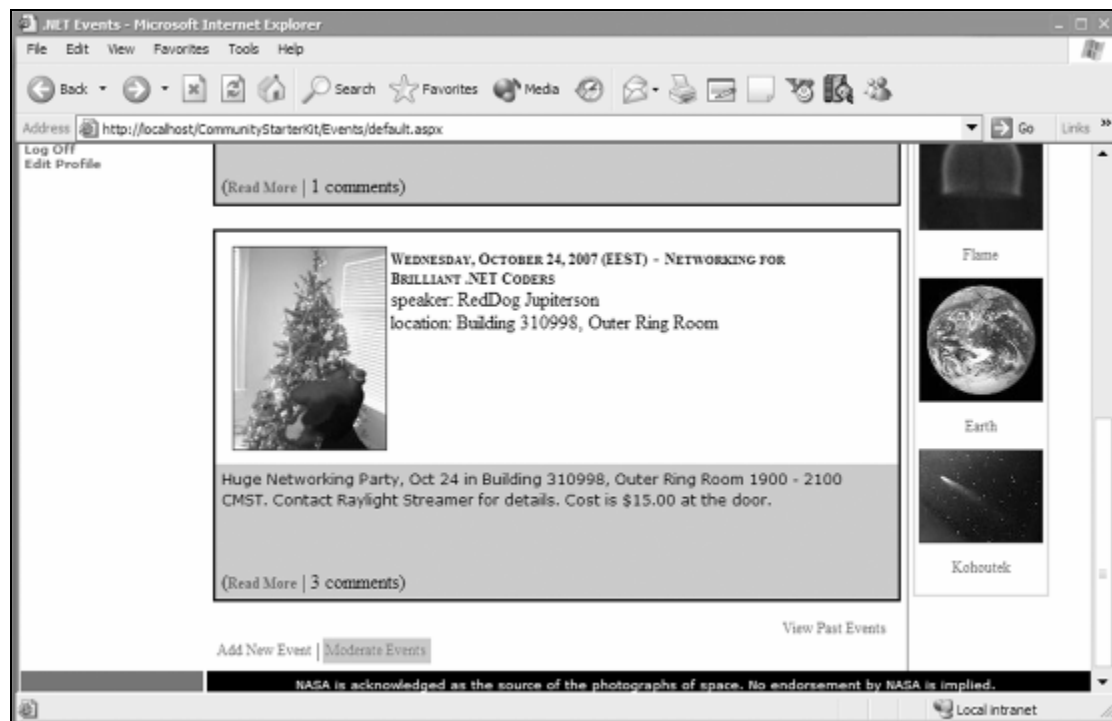


Moderation allows you to enable or disable moderation for the website section and choose the roles that can act as moderators. When moderation is enabled, new content is not displayed until it has been approved by a moderator. By default, every CSK community has a system role named Community-Moderators that can be used for moderation. Please review Chapter 2 for more details.



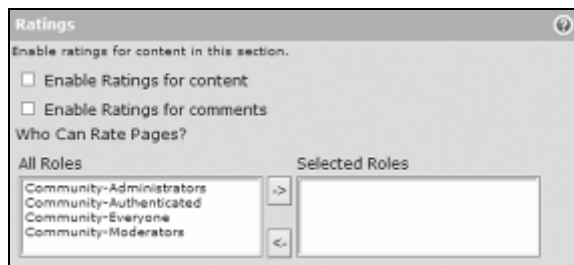
When adding content to a moderated section, the user is notified that their message will not appear straight away. The message they receive can be configured from Community Administration Menu | Edit Messages, and by default is This section is moderated. Please allow 24 hours for your content to appear.

A user assigned one of the selected roles will see the Moderate link that that appears at the bottom of a moderated community section. This link is altered depending on the section being browsed. It's called Moderate Articles for the Articles (Mission) section, Moderate Books for the Books section, and so on. The following screenshot shows the Moderate Events link:



After clicking the Moderate Events link, the moderator is shown a list of newly posted messages that they can then approve for publication.

The Ratings feature was explained in Chapter 2. This feature allows you to enable ratings for the content or the comments in the section. You can also specify roles that are allowed to rate content or comments.



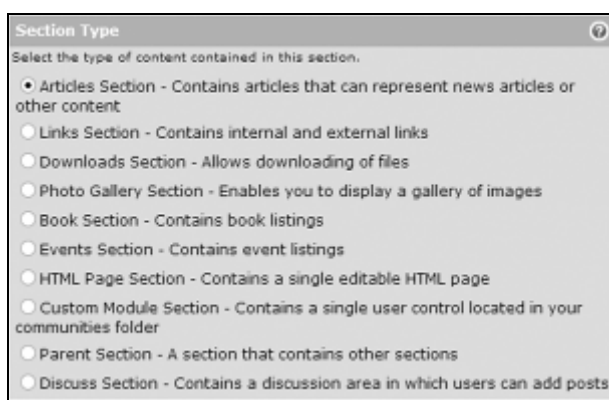
When the Enable Ratings for content checkbox is enabled, each article in a section receives an independent rating. On checking Enable Ratings for comments, a similar rating system is enabled for individual comments in the section.

Add New Section

Here you can add new content sections to the community website.

The Add Section form contains the same tabs as the Edit Section page: General, Appearance, Web Boxes, Security, Web Services, and Advanced. Apart from General, they all have the same functionality as their Edit Section counterparts. The only difference is that all these pages have an Add Section button at the bottom instead of Apply Changes.

The single new detail of this section is the additional Section Type field in the General tab where you select the type of the new section. We'll learn how to create a new community from scratch and practice creating new sections in Chapter 5:



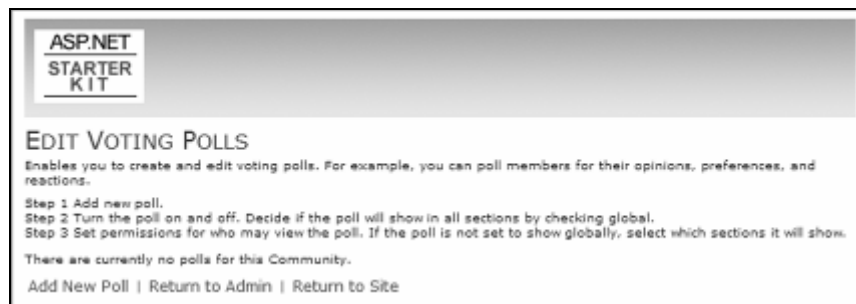
Section Type ?

Select the type of content contained in this section.

- ☒ Articles Section - Contains articles that can represent news articles or other content
- ☐ Links Section - Contains internal and external links
- ☐ Downloads Section - Allows downloading of files
- ☐ Photo Gallery Section - Enables you to display a gallery of images
- ☐ Book Section - Contains book listings
- ☐ Events Section - Contains event listings
- ☐ HTML Page Section - Contains a single editable HTML page
- ☐ Custom Module Section - Contains a single user control located in your communities folder
- ☐ Parent Section - A section that contains other sections
- ☐ Discuss Section - Contains a discussion area in which users can add posts

Edit Voting Polls

This page allows you to add new voting polls or modify existing ones. By default, no voting polls are created when the CSK is installed. The following screenshot shows the Edit Voting Polls page:



ASP.NET STARTER KIT

EDIT VOTING POLLS

Enables you to create and edit voting polls. For example, you can poll members for their opinions, preferences, and reactions.

Step 1 Add new poll.
 Step 2 Turn the poll on and off. Decide if the poll will show in all sections by checking global.
 Step 3 Set permissions for who may view the poll. If the poll is not set to show globally, select which sections it will show.

There are currently no polls for this Community.

[Add New Poll](#) | [Return to Admin](#) | [Return to Site](#)

Clicking on the Add New Poll link leads you to the Edit/Add Voting Poll page:

EDIT/ADD VOTING POLL																																																									
Active?	☒ Check this box when you are ready for this poll to go live.																																																								
Question:	Do you like our new website?																																																								
Global?	☒ Check this box if you want this poll to be available to all Sections.																																																								
Date Range: (Not Required)	Start Date: 3/1/2004 [Show Calendar] End Date: 3/7/2004 <table border="1"> <thead> <tr> <th colspan="7">< March 2004 ></th> </tr> <tr> <th>Sun</th> <th>Mon</th> <th>Tue</th> <th>Wed</th> <th>Thu</th> <th>Fri</th> <th>Sat</th> </tr> </thead> <tbody> <tr> <td>29</td> <td>1</td> <td>2</td> <td>3</td> <td>4</td> <td>5</td> <td>6</td> </tr> <tr> <td>7</td> <td>8</td> <td>9</td> <td>10</td> <td>11</td> <td>12</td> <td>13</td> </tr> <tr> <td>14</td> <td>15</td> <td>16</td> <td>17</td> <td>18</td> <td>19</td> <td>20</td> </tr> <tr> <td>21</td> <td>22</td> <td>23</td> <td>24</td> <td>25</td> <td>26</td> <td>27</td> </tr> <tr> <td>28</td> <td>29</td> <td>30</td> <td>31</td> <td>1</td> <td>2</td> <td>3</td> </tr> <tr> <td>4</td> <td>5</td> <td>6</td> <td>7</td> <td>8</td> <td>9</td> <td>10</td> </tr> </tbody> </table> [Hide Calendar]	< March 2004 >							Sun	Mon	Tue	Wed	Thu	Fri	Sat	29	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1	2	3	4	5	6	7	8	9	10
< March 2004 >																																																									
Sun	Mon	Tue	Wed	Thu	Fri	Sat																																																			
29	1	2	3	4	5	6																																																			
7	8	9	10	11	12	13																																																			
14	15	16	17	18	19	20																																																			
21	22	23	24	25	26	27																																																			
28	29	30	31	1	2	3																																																			
4	5	6	7	8	9	10																																																			
Display Results?	☐ Check this box if you want the results displayed.																																																								
Save Cancel																																																									
Return to Edit Polls Return to Admin Return to Site																																																									

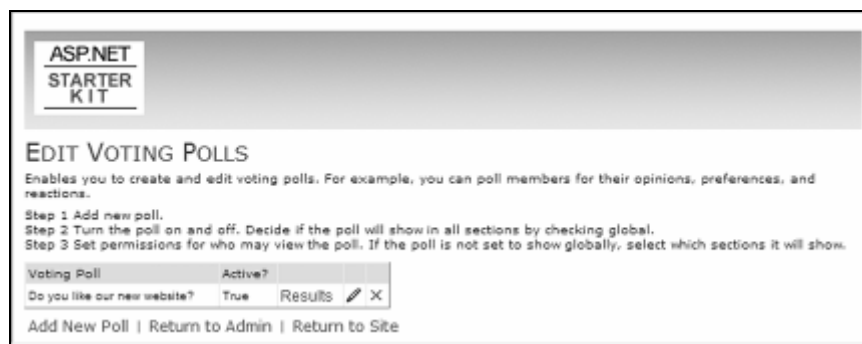
The voting poll will be shown to the visitors only if the Active checkbox is checked and the current date is within the specified Date Range. If a Date Range is not specified, the visit poll visibility will only be determined by the Active checkbox.

A global voting poll will be displayed in all sections of the site. If the voting poll is not global, the administrator can choose (in the next screen) the sections where it should be displayed:

EDIT/ADD VOTING POLL													
Active?	☒ Check this box when you are ready for this poll to go live.												
Question:	Do you like our new website?												
Global?	☒ Check this box if you want this poll to be available to all Sections.												
Date Range: (Not Required)	Start Date: 3/1/2004 [Show Calendar] End Date: 3/7/2004 [Show Calendar]												
Display Results?	☐ Check this box if you want the results displayed.												
Poll Choices:	Yes X up down No X up down Add Choice												
Roles:	<table border="0"> <tr> <td>All Roles</td> <td>Selected Roles</td> </tr> <tr> <td>Community-Administrators</td> <td>-> Community-Everyone</td> </tr> <tr> <td>Community-Authenticated</td> <td></td> </tr> <tr> <td>Community-Moderators</td> <td><-</td> </tr> </table>	All Roles	Selected Roles	Community-Administrators	-> Community-Everyone	Community-Authenticated		Community-Moderators	<-				
All Roles	Selected Roles												
Community-Administrators	-> Community-Everyone												
Community-Authenticated													
Community-Moderators	<-												
Sections: (Ignored if "Global" is checked)	<table border="0"> <tr> <td>All Sections</td> <td>Selected Sections</td> </tr> <tr> <td>Home</td> <td></td> </tr> <tr> <td>Mission</td> <td></td> </tr> <tr> <td>Events</td> <td></td> </tr> <tr> <td>.NET Books</td> <td></td> </tr> <tr> <td>Favorite Links</td> <td></td> </tr> </table>	All Sections	Selected Sections	Home		Mission		Events		.NET Books		Favorite Links	
All Sections	Selected Sections												
Home													
Mission													
Events													
.NET Books													
Favorite Links													
Save Cancel													
Return to Edit Polls Return to Admin Return to Site													

After clicking the Save button, a more detailed form with voting poll choices appears. The most important new detail is the ability to choose the poll choices. Also, you can select the roles that can view the poll; for non-global polls, you can select the sections where the poll is active.

After saving the new poll details, the poll is listed in the Community Administration Menu | Edit Voting Polls page, where you can see the results, edit its options, or delete it:

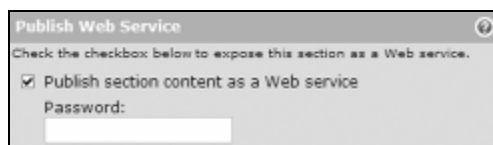


Edit Service Subscriptions

Through this menu you can subscribe to content from other CSK websites. For example, if your community has an Articles section, you can populate it by duplicating the contents from the Articles section in another community.

Towards the end of this chapter, you'll learn how to create new communities on the same CSK installation. For now, we'll do a little trick and use the same community to expose and consume web services data. However, the same steps would be required if we had two distinct communities.

First, in the community that exposes the information, you need to create a web service for one of the existing sections. Edit your Books section, go to the Web Services menu, and check Publish section content as a Web service:



That's all you have to do on the community exposing the web service! The web service address to access this section will be the same as its web address, except that it ends with `asmx` instead of `aspx`. In our example, the address of the newly created web service is `http://localhost/CommunityStarterKit/.NET+Books/default.aspx`.

Now, let's consume this web service. Go to Community Administration Menu | Edit Service Subscriptions in any CSK website where you have administration rights. On that page, click the Add New Service Subscription link. Choose a name for the service and type in its address:

ASP.NET STARTER KIT

ADD NEW SERVICE SUBSCRIPTION

On this page you can add a new subscription to a remote community section.

The URL is the address of the remote section. For example, to retrieve content from a remote community section located at <http://www.someSite.com/Events/Default.aspx>, you would enter the URL <http://www.someSite.com/Events/Service.aspx>.

Service Name:

Service URL:

Password:

[Return to Edit Services](#) | [Return to Admin](#) | [Return to Site](#)

The service will get listed in the Edit Service Subscriptions page:

ASP.NET STARTER KIT

EDIT SERVICE SUBSCRIPTIONS

You can create service subscriptions to any type of section except for Downloads and Photo Gallery sections. A service subscription enables you to retrieve content from other community Web sites. When you create a service subscription, content is automatically retrieved from another community Web site through an XML Web service. For example, if your community has an articles section, then you can retrieve articles from another community and add them to your articles section.

To create a new service subscription, you need to supply the URL of the remote community section that contains the content that you want to retrieve. The remote section must be exposed as a Web service.

You can monitor a service subscription through the Activity log. Every time content is retrieved from the remote community, an entry is added to the Activity log.

Refresh	Edit	Delete	Name	Url	Refresh Rate
			Remote Books	http://localhost/CommunityStarterKit/.NET+Books/default.aspx	Every 15 minute(s)

[Add New Service Subscription](#) | [Return to Admin](#) | [Return to Site](#)

After creating this service subscription, you need to create a new section in the remote community website to retrieve data from the registered web service. The following screenshot demonstrates how this is done:

ASP.NET
STARTER
KIT

ADD SECTION

Use this page to add a new section to your community. Click one of the tabs to modify a particular type of property.

General | Appearance | Web Boxes | Security | Web Services | Advanced

Section Name ⓘ
The name of the section is used to create the section path.
Remote Books

Section Title ⓘ
The title of the section is displayed in the browser title bar and the body of the page.
Remote Books

Section Menu Title ⓘ
The section menu title is displayed in menus.
Remote Books

Section Description ⓘ
The section description is displayed when the section is listed in a parent page.
Excellent books recommended by our colleagues from the Remote community!

Section Type ⓘ
Select the type of content contained in this section.

- ☐ Articles Section - Contains articles that can represent news articles or other content
- ☐ Links Section - Contains internal and external links
- ☐ Downloads Section - Allows downloading of files
- ☐ Photo Gallery Section - Enables you to display a gallery of images
- ☒ Book Section - Contains book listings
- ☐ Events Section - Contains event listings
- ☐ HTML Page Section - Contains a single editable HTML page
- ☐ Custom Module Section - Contains a single user control located in your communities folder
- ☐ Parent Section - A section that contains other sections
- ☐ Discuss Section - Contains a discussion area in which users can add posts

Parent Section ⓘ
Select a parent section for this section.
Home ▼

Section Enabled ⓘ
Uncheck to hide this section.
☒

Add Section

[Return to Edit Sections](#) | [Return to Admin](#) | [Return to Site](#)

After entering the necessary information, click on the Web Services link and add Remote Books to the Selected Items list. This will ensure that the section we're creating will import its data from the web service. After this, click on the Add Section button:

ASP.NET
STARTER
KIT

ADD SECTION

Use this page to add a new section to your community. Click one of the tabs to modify a particular type of property.
General | Appearance | Web Boxes | Security | Web Services | Advanced

Publish Web Service

Check the checkbox below to expose this section as a Web service.

☐ Publish section content as a Web service

Password:

Subscribe to Web Services

Web Services retrieve content retrieved from other Web sites.

All Items

Selected Items

Remote Books

->

<-

Add Section

Return to Edit Sections | Return to Admin | Return to Site

Once the data from the web service is refreshed (in about 15 minutes by default), you'll be able to see the new books section populated in the new Subscribed Books section of your community. Until then, you'll still be able to see and access the new section, but it will appear empty.

You can edit the Web Services update interval by modifying the `CreateTimers` method in `Global.asax` and then recompiling the project. You'll learn more about this file in Chapter 6.

You can monitor web service activity from Community Administration Menu | View Activity Log.

Edit Messages

The Edit Messages menu allows you to edit the Email Notification, Moderation, and Password Reminder messages. The notification and the password reminder are sent by e-mail; the moderation message is shown after the user posts a message to a moderated section.

You need to use the drop-down list to select the message to edit, amend the Message Body, and then press Edit Message to save the changes:

**ASP.NET
STARTER
KIT**

MESSAGES

In this section, you can create messages, which can be used as error or general information text in response to an action by a user. For example, 'Invalid Password' or 'Please allow 24 hours for your submission to appear.'

Email Notification ▼

This email message is sent to users when they receive a notification

Message Title
Notification: <ContentTitle>

Message Body
Dear <Username>,

A new item was added to the <SectionName> section. You can view this item by clicking the following link:

 <ContentLink>

If you would like to disable these notifications, you can edit your profile at <EditProfileLink>

Edit Message

[Return to Admin](#) | [Return to Site](#)

You can use the following special tags for the Email notification. The tags themselves are quite self-explanatory:

- <Username>
- <EditProfileLink>
- <SectionName>
- <ContentLink>
- <ContentTitle>

For the password reminder, you can use the following special tags:

- <Username>
- <Password>
- <FirstName>
- <LastName>
- <FullName>
- <EditProfileLink>
- <HomeLink>

Edit Topics

Earlier, you learned how to associate community content with topics in Community Administration Menu | Edit Sections | Edit Section | Advanced | Topics. Through the Edit Topics menu, you can edit the list of available topics and choose whether you want to enable the topic menu or not.

You can choose to use topics individually for every community section. You can edit the general list of topics and enable or disable the Topic Menu from Community Administration Menu | Edit Topics.

The Edit Topics page welcomes you with the current topic settings:

ASP.NET STARTER KIT

EDIT TOPICS

Topics enable you to categorize content across different sections. The topic menu provides an alternative to the section menu for navigating your community.

The topic menu automatically displays the last 5 distinct topics associated with content items added to your community. The topic menu is automatically refreshed every 10 minutes.

☒ Enable Topic Menu?

Edit	Delete	Topic
	X	Internet
	X	Politics

[Add New Topic](#) | [Return to Admin](#) | [Return to Site](#)

When adding a new topic (as shown in the following screenshot), you can set its name, description, and (optionally) a topic image. If an image is chosen, it will appear in the topic menu instead of the topic name. Adding the topic image is constrained by the Community Quota settings.

ASP.NET STARTER KIT

ADD NEW TOPIC

On this page you can add a new Topic. You can create a topic by providing only a topic name and description or by providing a topic name, image, and description.

User Quota: 10,000 KB	User Quota Used: 2,088 KB
Community Quota: 10,000 KB	Community Quota Used: 2,480 KB

Topic Name:

Topic Image: (optional) [Browse...](#)

Topic Description:

[Add Topic](#)

[Return to Edit Topics](#) | [Return to Admin](#) | [Return to Site](#)

Edit Users

This section allows the community administrator to see the list of registered users as well as edit their properties and security roles. However, there are two limitations:

- You cannot add new users through this menu. To create a new account, the user needs to use the Register page.
- You cannot delete users through this menu. The CSK doesn't have functionality to delete users. A programmer posted a solution to this problem on the ASP.NET CSK forums at <http://www.asp.net/Forums/ShowPost.aspx?tabindex=1&PostID=187894>.

The Edit Users page looks like this:

ASP.NET STARTER KIT

EDIT USERS

The Edit Users page allows you to add, edit, delete, or view information about the users in your community. You can view all users at once, or alphabetically by letter. Furthermore, you can sort your results by username, the user's full name, or by the date joined. To edit a user, click on the desired username.

From this page, you can also create, edit, or delete roles for your users. You can use roles to control who can view, add, edit, delete, moderate, or comment on content.

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | All

Sort By: Order By:

Username	Full Name	Date Joined
Admin	Admin User, Admin User	2/5/2004
AnC899	C899, An	1/9/2003

[Edit Roles](#) | [Return to Admin](#) | [Return to Site](#)

When clicking on a username, the administrator is forwarded to a page where they can edit any of the user's details, including the security roles and database quota:

EDIT USERS

On this page, you can edit a user's profile and add or remove roles assigned to the user.

Required Information

Username:

Password:

Password Again:

Email Address:

First Name:

Last Name:

The user's password is retrieved from the database and shown in a textbox with the `TextMode` property set to `password`. This "hides" the entered text, but the password is actually written in the HTML source in clear text. This makes it a useless feature because the user's privacy is not effectively protected.

When clicking the [Edit Roles](#) link, the administrator is shown the list of current security roles and provided with the means to add new roles. The two system roles, `Community-Administrators` and `Community-Moderators`, cannot be removed from the list of roles, unlike the roles added by the administrator.

Also, bear in mind that when setting security permissions for other parts of the site, you have two additional roles to choose from: `Community-Everyone` (which refers to any visitors, including anonymous visitors) and `Community-Authenticated` (which refers to authenticated users). The following screenshot shows the [Add or Edit Roles](#) page:



ASP.NET STARTER KIT

ADD OR EDIT ROLES

On this page you can create, edit, or delete roles for your users. Roles can be used to prohibit all but selected users from secure areas in your community. For example, only users in the `Community-Administrators` role have access to these administration pages. Existing roles and descriptions of each role are available in the table below. System roles cannot be deleted from the community. Any other roles can be added, edited, or deleted from this page.

Role	Description	System
<code>Community-Administrators</code>	Community Administrators role.	True
<code>Community-Moderators</code>	Community moderators.	True

Add Role

Role Name:

Role Description:

[Return to Edit Users](#) | [Return to Admin](#) | [Return to Site](#)

Edit Web Box Services

Here you can manage the list of services that can be used to populate the Web Box services in your community website. You learned earlier in this chapter about Web Box services when discussing the [Community Administration Menu](#) | [Edit Sections](#) | [Edit Section](#) | [Web Boxes](#) | [Web Box Services](#) menu.

Remember that the CSK allows you to subscribe to two kinds of external data sources (web services exposed by other CSK websites or RSS sources), but can expose content only through web services.

By default, the Lunar .NET community has a single RSS source that gathers information from the official Star Wars page:

ASP.NET STARTER KIT

EDIT WEB BOX SERVICES

You can use Web Box Services to retrieve information from other Web sites and display the information in your community Web pages. Web Box Services include both Community Web services and RSS services.

Refresh	Edit	Delete	Name	Uri	Maximum Items	Refresh Rate	Last Refreshed
			Star Wars	http://www.starwars.com/community/webmasters/starwars.rdf	5	Every 1 hour (s)	2/7/2004 4:38:07 PM

[Add New Service](#) | [Return to Admin](#) | [Return to Site](#)

In Community Administrator Menu | Edit Service Subscriptions, click Add New Service to add the Books section that you exposed earlier as a Web Box service:

ASP.NET STARTER KIT

ADD NEW SERVICE

On this page you can add a new Web Box Service.

Service Name:

Service URL:

Service Type:

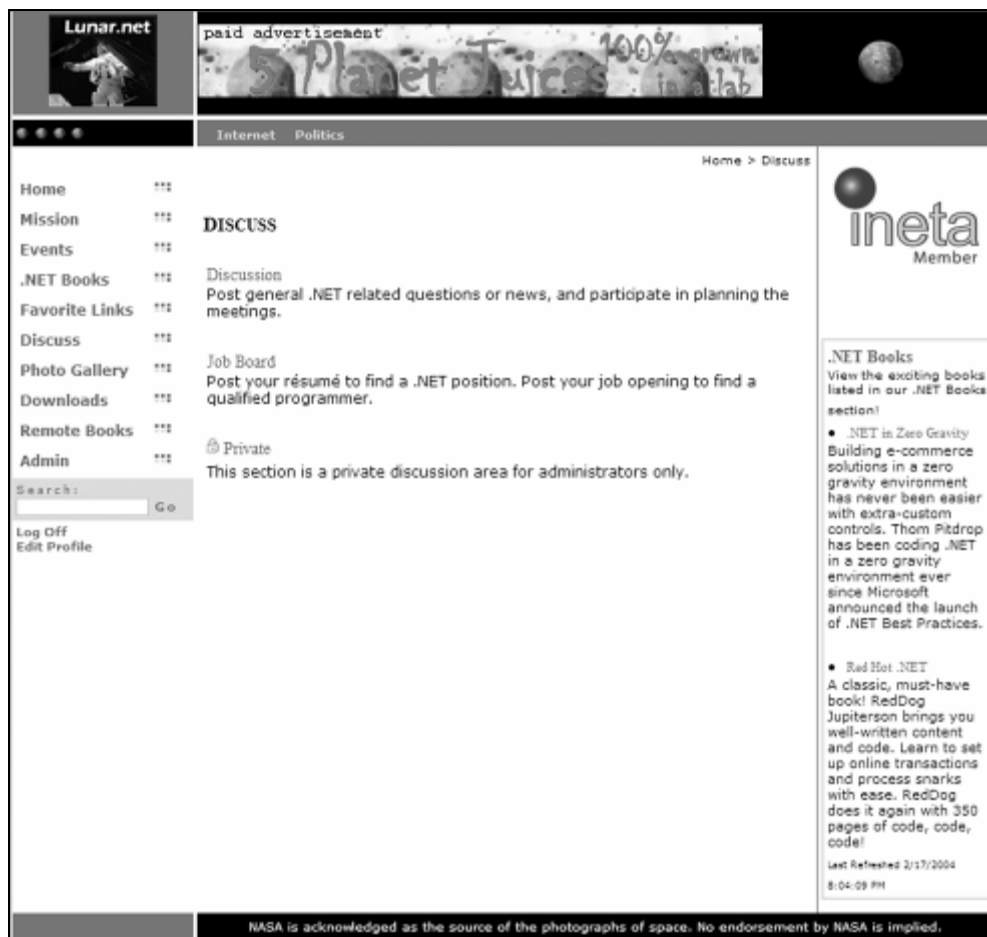
Refresh Rate:

Maximum Items:

Password:

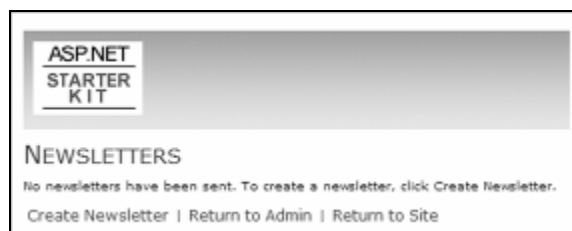
[Return to Edit Services](#) | [Return to Admin](#) | [Return to Site](#)

After adding the service here, it will become available in the list of data sources for Web Boxes, which appears when editing a section at Community Administration Menu | Edit Sections | Edit Section | Web Boxes, or when adding a new section. The following screenshot shows a Web Box populated from our own service:



Send Newsletter

Through this option, you can send a newsletter to all the users in your community that have chosen to receive newsletters by e-mail:



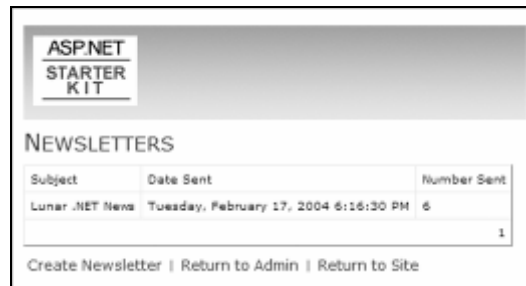
After clicking Create Newsletter, you're presented with the form that has the option to use e-mail templates, a very useful feature for the administrator. You also have the option to choose between HTML and plain-text newsletters; you can preview the newsletter and even send a test e-mail.

You can use the following tags in the e-mail body. They will be substituted automatically using real values:

Tag	Description
<Username>	The user's registered username
<Password>	The user's registered password
<FirstName>	The user's first name
<LastName>	The user's last name
<FullName>	The user's full name including first and last name
<EditProfileLink>	The link to the user's Edit Profile page
<HomeLink>	The link to the home page of the community
<NewArticles>	A list of the newest articles posted to the community
<NewBooks>	A list of the newest books posted to the community
<NewEvents>	A list of the newest events posted to the community

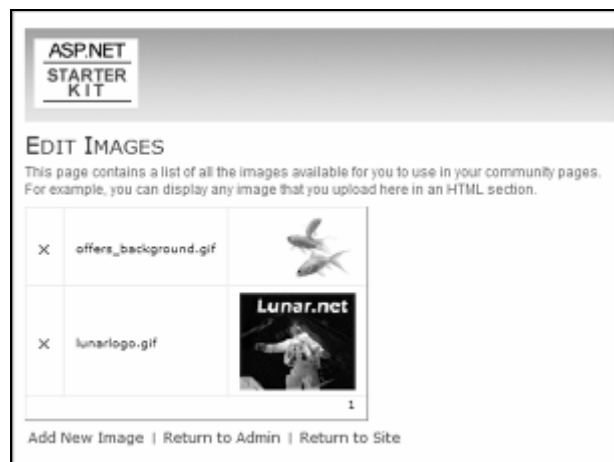
Tag	Description
<NewLinks>	A list of the newest links posted to the community
<NewDownloads>	A list of the newest downloads posted to the community
<NewPhotos>	A list of the newest photos posted to the community
<NewDiscuss>	A list of the newest Discuss posts added to the community

After sending the newsletter, it is shown on the main NEWSLETTERS page:



Edit Images

This section displays the images that are currently stored in the database as community images (such as the community logo image). Images that appear inside separate sections (such as the ones in Photo Gallery or Book sections) are stored separately in the database and do not show up here. The Edit Images page looks like in the following picture:



You need to access this page if you want, for example, to remove the image that you added as a section logo at Community Administration Menu | Edit Sections | Edit Section | Appearance, because that menu and other similar menus don't have the option to remove images.

You'll learn in Chapter 6 the details about how CSK stores some of the image files in the database (even though they are given names, like those of "real" files), and how to store image files directly in the Windows file system. Until then, here are a few useful details:

- Images that appear in the Books, Events, and Photo Gallery sections are stored in the `Community_CommunityImages` table of the CSK database (the default name for the database, which can be changed at setup, is `CommunityStarterKit`). These images do not show up in the Edit Images page of the Community Administration Menu.
- The other images that you can upload to the website but don't get displayed inside a particular section, such as the community logo, or images that appear in the topics bar are stored in the `Community_SectionImages` table. These are the images you can manage through the Community Administration Menu.
- The classes that handle image-related operations are located in `Engine/Framework/Images/Components`. `ImageUtility` contains functions that manipulate database images, and `ImageHandler` is the class that resolves requests to image files by retrieving them from the database.
- If you add pages that reference "real" image files stored in the Windows file system, you need to store them in `Communities\Common\Images` or in another folder that has a `web.config` file, which unhooks the image file requests from the special CSK image-file handler.

View Activity Log

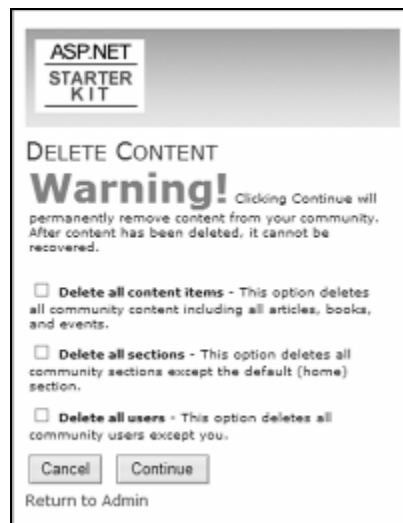
The Activity Log displays the last 50 events that happened on your website. The following screenshot shows a typical Activity Log:

ACTIVITY LOG	
The Activity log displays the last 50 events that occurred in your community. You can use the Activity log to track errors and other events.	
Entry	Date
Error retrieving remote content	Monday, February 16, 2004 7:37 PM
Retrieving content from http://localhost/CommunityStarterKit/.NET+Books/default.aspx	Monday, February 16, 2004 5:55 PM
Received service subscription request from Community1	Monday, February 16, 2004 5:55 PM
Retrieving content from http://localhost/CommunityStarterKit/.NET+Books/default.aspx	Monday, February 16, 2004 5:19 PM
Received service subscription request from Community1	Monday, February 16, 2004 5:19 PM
Added 2 pages to the Remote Books section	Monday, February 16, 2004 5:04 PM
Added 2 pages to the Remote Books section	Monday, February 16, 2004 5:04 PM
Retrieving content from http://localhost/CommunityStarterKit/.NET+Books/default.aspx	Monday, February 16, 2004 5:04 PM
Received service subscription request from Community1	Monday, February 16, 2004 5:04 PM
Clear Entries Return to Admin Return to Site	

Each entry can be clicked to learn additional information about the event. By default, the Activity Log records only events related to web services. Changing what the activity log records requires modifying the CSK code, and cannot be controlled through the Community Administration Menu.

Delete Content

This menu allows you to remove community content from the CSK database. You have the option to delete all community sections, community content, and community users, except the logged in user (usually Admin):



The screenshot shows a web-based dialog box titled "DELETE CONTENT Warning!". At the top left is the "ASP.NET STARTER KIT" logo. The main heading is "DELETE CONTENT" followed by "Warning!". Below this, a warning message states: "Clicking Continue will permanently remove content from your community. After content has been deleted, it cannot be recovered." There are three checkboxes with labels: "Delete all content items - This option deletes all community content including all articles, books, and events.", "Delete all sections - This option deletes all community sections except the default (home) section.", and "Delete all users - This option deletes all community users except you." At the bottom, there are "Cancel" and "Continue" buttons, and a link that says "Return to Admin".

View Documentation

Through this menu, you access the CSK documentation. However, not all the documentation is directly accessible through this menu and you can find more in the documentation folder within your installation directory. In particular, there is information on installing the CSK for an ISP.

Administering Advertisements

The two advertisement boxes are controlled by AdRotator controls. The AdRotator control (which is part of the .NET Framework and not CSK specific) is configured through external XML files. Please consult the AdRotator Web Server Control in the .NET Framework General Reference for a description of the XML configuration files.

In the CSK, the AdRotator control at the top of the page is configured through `/Advertisements/BannerAds.xml`, and the AdRotator on the right is configured with `/Advertisements/BoxAds.xml`.

The Community Administration Menu doesn't provide the functionality to edit the advertisements to be displayed. You'll need to manually edit the two XML configuration files to manage the displayed ads.

The source code for BannerAds.xml is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>

  <Ad>
    <ImageUrl>juice_banner.gif</ImageUrl>
    <NavigateUrl></NavigateUrl>
    <AlternateText>Visit our sponsor Five Planet Juices</AlternateText>
    <Impressions>50</Impressions>
  </Ad>

  <Ad>
    <ImageUrl>grannyseggs_banner.gif</ImageUrl>
    <NavigateUrl></NavigateUrl>
    <AlternateText>Visit our sponsor Granny's Eggs</AlternateText>
    <Impressions>50</Impressions>
  </Ad>

</Advertisements>
```

Administering Quotes

You're probably wondering how to change the quotes that appear in the upper-right Web Box since the Community Administration Menu doesn't provide this feature. Let's solve this mystery here.

The ASCX controls that populate the Web Boxes are located in the Communities/Common/WebBoxes folder. The file that generates the quotes is named Quote.ascx. Opening this file reveals the source of these quotes:

```
<%@ Control Language="C#" %>
<script runat="server">
  void Page_Load()
  {
    ArrayList col = new ArrayList();
    col.Add("Service is the rent we pay for being. It is the very purpose of
      life, and not something you do in your spare time. <br> --- Marion
      Wright Edelman");
    col.Add("You must be the change you wish to see in the world. <br> ---
      Mahatma Ghandi");
    col.Add("Make others happy and joyful. Your happiness will multiply a
      thousand fold. <br> --- Swami Sivananda");
    col.Add("The influence of each human being on others in this life is a kind
      of immortality. <br> --- John Quincy Adams");
    col.Add("Love sought is good, but given unsought is better. <br> ---
      Shakespeare");
    col.Add("Here is a test to find out whether your mission in life is complete.
      If you're alive, it isn't. <br> --- Richard Bach");
    col.Add("There is no such thing in anyone's life as an unimportant day. <br>
      --- Alexander Woollcott");
```

```
col.Add("How far you go in life depends on your being tender with the young,  
compassionate with the aged, sympathetic with the striving and  
tolerant of the weak and the strong. Because someday in life you  
will have been all of these. <br> --- George Washington Carver");  
col.Add("People rarely succeed unless they have fun in what they are doing.  
<br> --- Dale Carnegie");  
col.Add("Sit on a baby's bib and SPIT HAPPENS <br> --- Anonymous");  
col.Add("May the smile on your face Come straight from your heart <br> ---  
Anonymous");  
col.Add("Most good judgment comes from experience. Most experience comes from  
bad judgment. <br> --- Anonymous");  
col.Add("The true \"final frontier\" is in the minds and the will of people.  
<br> -- Gen. Michael E. Ryan, U.S. Air Force Chief of Staff");  
col.Add("I don't pretend to understand the Universe - it's a great deal  
bigger than I am. <br> -- Thomas Carlyle");  
col.Add("When we try to pick out anything else in the Universe, we find it  
hitched to everything else in the Universe. <br> -- John Muir");  
col.Add("Not all who wander are lost. <br> -- Tolkien");  
  
Random objRan = new Random();  
lblQuote.Text = (string)col[objRan.Next(col.Count)];  
}  
</script>  
<span class="WebBox_TitleRow"><asp:Label id="lblQuote" Runat="Server" /></span>
```

There's an `ArrayList` object that is manually filled with quotes, and the one to be displayed is chosen using the `Random` class. Having the list of quotes directly inside the ASCX file and not in an external configuration file or the database may not be a particularly exciting solution, but it's simple and effective.

You'll need to edit `Quote.ascx` in order to add, remove, or change the quotes. You don't need to restart the CSK or recompile the project after making the change.

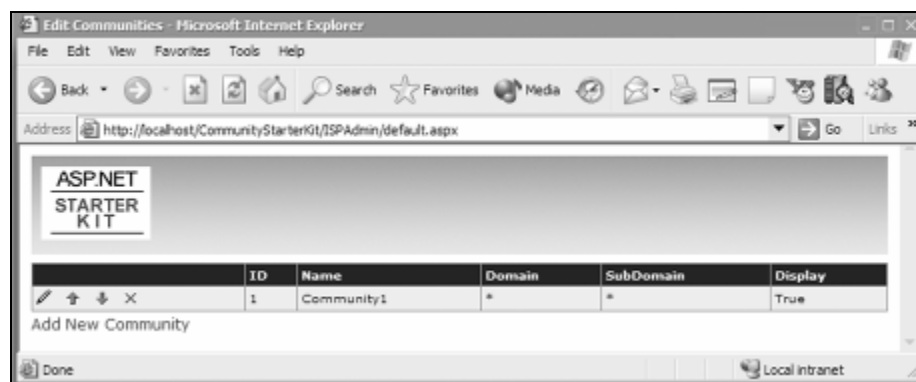
The ISP Administration Page

You can host multiple communities using a single CSK installation. The ISP (Internet Service Provider) Administration page allows the ISP Administrator to manage the communities that coexist on a CSK installation. One of the options you can set here is the database quota for each community.

The ISP Admin page can be found by default at `http://localhost/CommunityStarterKit/ISPAdmin`. At installation, you had the choice of changing its default administration username and password combination. By default the username is `ISPAdmin` and its password is `ISPPassword`. The ISP admin username and its password are stored in clear text in `Web.Config` (located in the CSK root folder). You'll see more about `Web.Config` in Chapter 6.



After logging in as an ISP Administrator, you get a list of installed communities. Here, you can delete or update information for existing communities or add new communities. By default, you have only one community (Community1), which is the Lunar .NET community created at installation:



Creating a New Community

Let's experiment and learn how to use a single CSK installation to host multiple communities.

In order to be able to access multiple communities on the local machine, you'll need to assign additional host names to your computer's loopback address (127.0.0.1). You can do this by modifying the Hosts file in the folder C:\Windows\system32\drivers\etc.

By default, it contains something like this:

```
# Copyright (c) 1993-1999 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#       102.54.94.97       rhino.acme.com       # source server
#       38.25.63.10      x.acme.com           # x client host
127.0.0.1        localhost
```

You need to add a line that associates an additional host name with the computer's loopback address. Let's assume you're creating a new community named HappyStudents. Add the following line to the Hosts file:

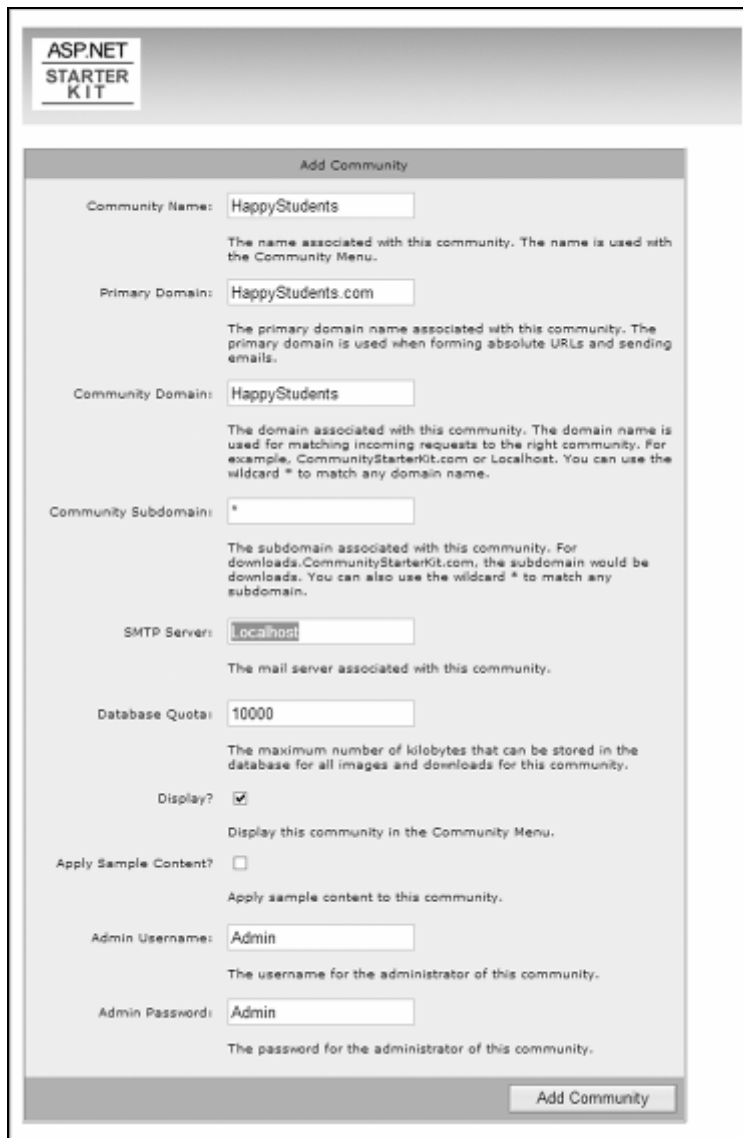
```
127.0.0.1        HappyStudents
```

Now, since HappyStudents redirects to the same location as localhost, accessing <http://HappyStudents/CommunityStarterKit> will also redirect to the CSK. If no community associated with the HappyStudents domain exists, you will be forwarded to the same Lunar .NET community that you've been playing with so far.

However, the CSK can be instructed to load different communities depending on the host name used to access it. We will experiment adding a new community named HappyStudents, which will be accessed through <http://HappyStudents/CommunityStarterKit>, and will be different than the community you access through <http://localhost/CommunityStarterKit> (even though both localhost and HappyStudents refer to the same physical machine).

To create a new community while in the ISP Administration menu, click the Add New Community link. You'll be asked to provide a number of details for your new community. The main elements to pay special attention to are Community Domain and Community Subdomain.

The screenshot that follows displays the options you need to set for creating a new community named HappyStudents running on the local machine:



The screenshot shows the 'Add Community' form in the ASP.NET Starter Kit. The form is titled 'Add Community' and contains several input fields with descriptive text below them. The fields are: Community Name (HappyStudents), Primary Domain (HappyStudents.com), Community Domain (HappyStudents), Community Subdomain (*), SMTP Server (localhost), Database Quota (10000), Display? (checked), Apply Sample Content? (unchecked), Admin Username (Admin), and Admin Password (Admin). An 'Add Community' button is at the bottom right.

ASP.NET
STARTER
KIT

Add Community

Community Name: HappyStudents
The name associated with this community. The name is used with the Community Menu.

Primary Domain: HappyStudents.com
The primary domain name associated with this community. The primary domain is used when forming absolute URLs and sending emails.

Community Domain: HappyStudents
The domain associated with this community. The domain name is used for matching incoming requests to the right community. For example, CommunityStarterKit.com or Localhost. You can use the wildcard * to match any domain name.

Community Subdomain: *
The subdomain associated with this community. For downloads, CommunityStarterKit.com, the subdomain would be downloads. You can also use the wildcard * to match any subdomain.

SMTP Server: localhost
The mail server associated with this community.

Database Quota: 10000
The maximum number of kilobytes that can be stored in the database for all images and downloads for this community.

Display? ☒
Display this community in the Community Menu.

Apply Sample Content? ☐
Apply sample content to this community.

Admin Username: Admin
The username for the administrator of this community.

Admin Password: Admin
The password for the administrator of this community.

Add Community

Let's look at the complete list of fields you need to fill.

Community Name is the name of your new community. It is only important for administrative purposes, because it represents the name used to distinguish the community in the Communities folder.

Primary Domain is the domain on which your community will reside. This setting isn't critical as it is used only when sending e-mails. For example, if the primary domain is HappyStudents.com, its

newsletters will be sent from `news@HappyStudents.com`. The primary domain is not used to distinguish communities.

Community Domain is the domain on which the new community runs. In our scenario, you should set this to `HappyStudents` so that the CSK will load the new community when accessed from `http://HappyStudents/CommunityStarterKit`. Of course, you're not limited to specifying local domains as you can choose an Internet domain such as `HappyStudents.com`.

Community Subdomain is the subdomain for your new community. For the `HappyStudents` scenario, leave this to the default value of `*`. If you want to have more communities on the same community domain, you'll need to assign a different subdomain to each of them. For example, you can create two different communities under the `Students.com` domain, named `Happy.Students.com` and `Campus.Students.com`.

In this case, the domain is `Students.com` and the subdomains are `Happy` and `Campus`; the two communities can be accessed through `http://Happy.Students.com/CommunityStarterKit` and `http://Campus.Students.com/CommunityStarterKit`.

SMTP Server is used to specify the SMTP server used to send e-mails.

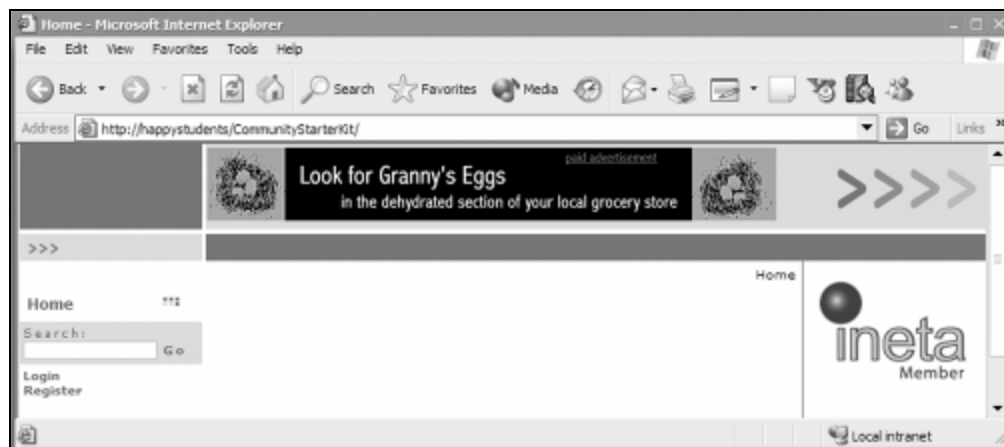
Database Quota is the limit you set for total number of photos and downloadable files that can be added to a community.

Display controls whether the community gets displayed in a control named `Community Menu`. This control is part of the CSK, but it is not used in any skin by default.

Add Sample Content controls whether the Lunar .NET sample community should be created by default when creating the new community. If you leave this unchecked, you'll start with an empty community. In Chapter 5, you will learn how to build a functional new community from scratch.

Admin Username and Admin Password are used to automatically create a new user with `Community-Administrator` privileges in the new community. This user will not have access to the ISP Admin pages.

After adding the new community, you'll access it through `http://HappyStudents/CommunityStarterKit`. If you checked `Apply Sample Content`, you'll be presented with the Lunar .NET community. Otherwise, you'll see an empty community as depicted in the following screenshot:



Summary

In this chapter, we have looked at the Community Administration Menu. This is a key tool for a Community-Administrator and the person building a new community. We have looked at using this tool to add and edit things such as website sections, voting polls, the web services that our site subscribes to, and user details. We have also seen some basic things that cannot be done through the tool, such as changing the quotes and also how to add new communities to the default installation.

After learning how to create skins in Chapter 4, we'll practice creating a new community from scratch using the tools we saw in this chapter.

4

Themes, Skins, and the Common Controls

In this chapter we'll talk about colors and layout. The colors of a web page are defined by the **style** being used, and the layout is generated by **skin files**. The style and the skin together form the **theme**, which defines the way each community page looks.

Having an appropriate theme is important because it reflects the personality of your community website. The Community Starter Kit (CSK) has been designed from the start to support easy integration of skins and styles, without needing to rebuild the project or change the database.

In this chapter you'll learn how to build personalized themes. You will:

- Learn about themes, skins, and styles and how they work
- Learn about the built-in CSK custom web controls
- Have your first look with the CSK Visual Studio .NET project (although we'll not do any programming using C# in this chapter)

This will prepare you for the next chapter where we'll build a new community website from scratch.

Meeting Themes, Skins, and Styles

A theme is composed of skins and styles. Skins are web user controls (files with the **.ascx** extension) containing a number of controls that generate the dynamic part of the page (such as information retrieved from the database).

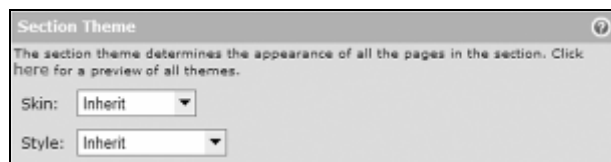
The following screenshot (Design view of Visual Studio .NET) shows how the skin file generating an article is built. You can see the skin file, which contains a number of controls (such as **Title**, **Author**, **BriefDescription**, and so on) that create the actual output:



Styles are standard **Cascading Style Sheets (CSS)** files. Simply said, a CSS file contains a set of rules that define how different page elements such as headers, links or normal text look. This book assumes you're already familiar with CSS, but in case you aren't, there are many tutorials on the Internet, such as the one at <http://www.w3.org/Markup/Guide/Style>.

Visual Studio .NET contains editors for both **.ascx** controls (skins) and **.css** files (styles).

In Chapter 3 you learned that you can easily change the theme of the community website using the Community Administration Menu and choosing the **Skin** and **Style**:



The CSK ships with the following skins: Arc, Cruise, Default, Eco, Frida, Library, Professional, Robotico; and the following styles: Arc, Cruise, Default, Default Funky, Default Lunar, Eco, Frida, Library, Library Graduate, and Professional (you can preview each of them by clicking the respective link). You can build a new theme by choosing any combination of a skin and style.

Apart from the existing skins and styles, for child sections you can choose the value **Inherit**, which means that the current section inherits the skin and style settings from its parent. This option is, of course, not available for the home page, which has no parent. You will usually set the theme for the home page in your community, and let all the other sections inherit these values.

Playing with Visual Studio

It's time you met the Visual Studio CSK project. You can find it at [Start | Programs | ASP.NET Starter Kits | ASP.NET Community Starter Kit \(CSVs\) | Community Starter Kit Source](#). Note that the first time you save the project, Visual Studio will ask you for a solution file name; then you can open the project using the solution file.

The **Solution Explorer** window in Visual Studio ([View | Solution Explorer](#)) will be your best friend in this chapter when browsing through the folder structure. The default themes are located under the **Communities/Common/Themes** folder. You'll notice that every theme has its own folder. The **Default** theme is, by far, the most complex of all. The other themes include only the elements that are different from the **Default** theme, and the missing elements are automatically taken from the **Default** theme.

The following screenshot shows a part of the CSK directory structure in the **Solution Explorer** window:



If the directory structure of the CSK looks intimidating, don't worry. As we analyze it you'll see it's not so complicated, after all.

Each theme folder has three subfolders: **Images**, **Skins**, and **Styles**.

Themes saved in the **Communities/Common/Themes** folder are visible to all communities of the CSK installation. For a theme to be visible to a specific community alone, it should be saved in the relevant **Communities/community_folder_name/Themes** folder.

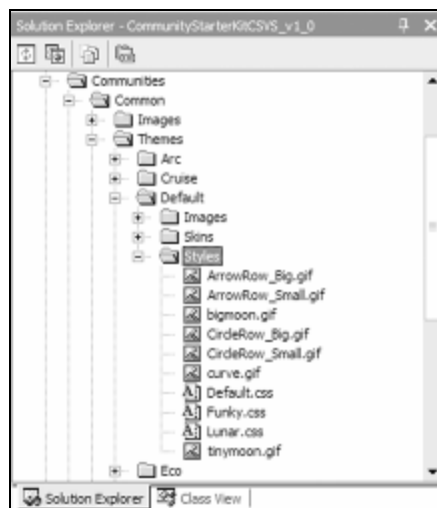
The **Images** folder contains image files used by the specified theme. Most image files shown in a community web page are retrieved from the database, including the community logo, the images that you can see in the Books, Events, Photo Gallery sections, and so on. Static images (that

cannot be dynamically added into the website, and are not configurable from the Community Administration Menu) are usually located directly in the file system for better performance. These folders should be configured within additional `Web.Config` files. Otherwise the CSK will try to retrieve them from the database and fail. We'll learn more about this process in Chapter 6.

The `Styles` folder contains one or more CSS files for the theme. The `Skins` folder has four subfolders corresponding to the four skin types that form the complete skin. In the following pages we'll first look at the styles, and then at how skins are composed.

Painting the Skin with Style

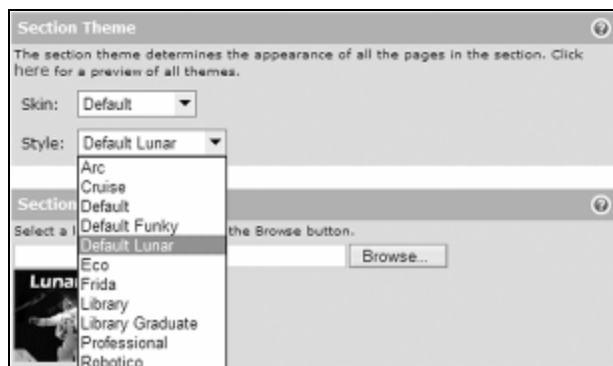
If the skin file decides *what* information to display, the style decides *how* to display it. If you look into the `Styles` subfolder of the Default skin, you can see that it has three separate styles (`Default.css`, `Funky.css`, and `Lunar.css`):



If you go to the [Community Administration Menu | Edit Sections | Edit Section | Appearance](#) section that we talked about in Chapter 3 for the main page, you'll see that the default theme is composed of the Default skin and the default Lunar style.

However, for any skin, you can choose any of the ten predefined styles, which are gathered from all the existing themes. Three of the styles (Default, Default Funky, and Default Lunar) are the ones defined for the `Default` theme (`Default.css`, `Funky.css`, `Lunar.css`).

Their names are automatically composed by appending the style name to the theme name, except for the `Default.css` style whose name is not appended (it would result in Default Default). Usually, you'll want to choose a skin and a style from the same theme, but you are free to apply any style to a skin and obtain a custom theme.



If you open one of the `.css` files in Visual Studio .NET, you'll see that they are very well documented. You'll know exactly what to modify if you want to build a new style or change an existing one.

What are Skins Made Of?

A CSK skin is formed of a large number of skin files (`ascx` Web User Controls). There are four kinds of skin files: **page skins**, **content skins**, **control skins**, and **template skins**. Each of them applies to a different part of a community web page.

We're dealing with two kinds of controls:

- **The Web User Controls:** Used in the skin files, these files have the `ASCX` extension (they begin with the `<% @Control %>` directive) and contain HTML code to compose the layout, and include references to ASP.NET controls (such as `TextBox`) and CSK Custom Controls.
- **The Custom controls:** They are .NET classes that inherit from `System.Web.UI.WebControls.WebControl` and generate HTML output when executed (we'll discuss these later in the chapter).

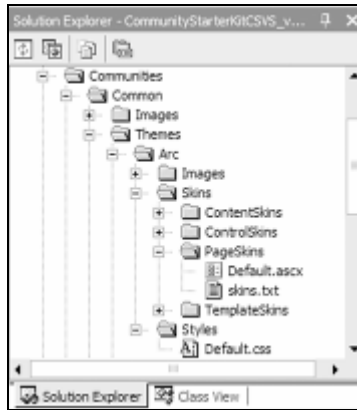
When developing a new CSK skin, you add controls to one or more subfolders in the `Skins` folder, which correspond to the four skin types: `ContentSkins`, `ControlSkins`, `PageSkins`, and `TemplateSkins`. Because CSK supports skin inheritance, when developing a new skin you need to develop only the individual items that are different from the default skin.

Default is the complete theme that contains skin elements for all website sections. The other themes only override parts of the elements in the `Default` theme.

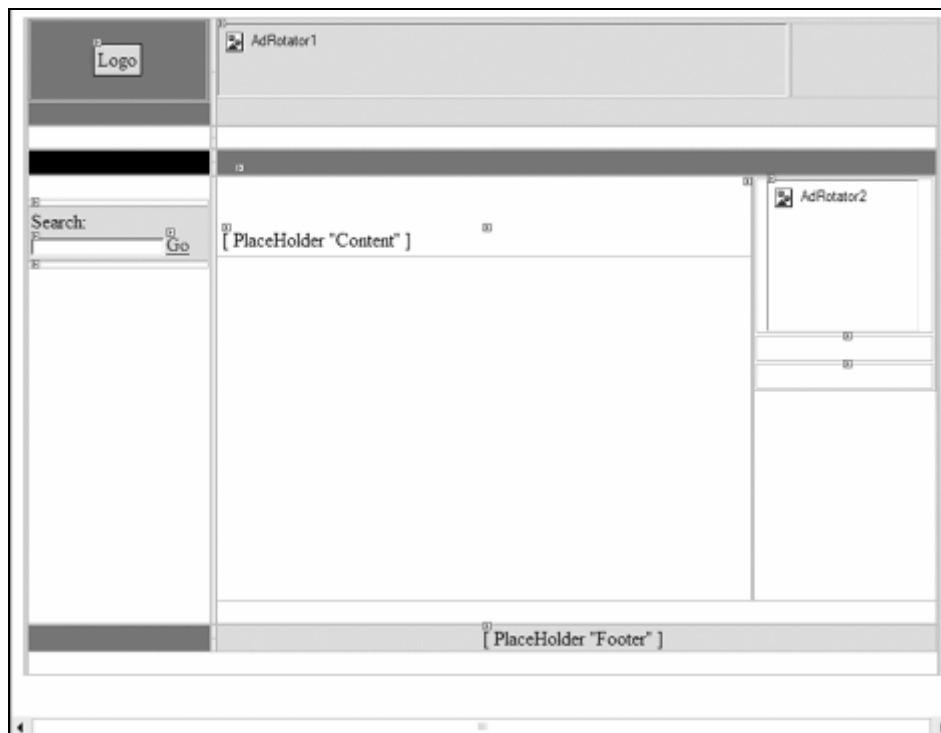
Page Skins

No matter what theme you choose, you'll notice that all community pages will have the same general structure such as the position of navigation menu, search box, login controls, banner ads, and so on.

This general structure is defined by the page skin. A theme can contain only one page skin. The page skin is a control that is always named **Default.ascx**:



The following screenshot shows the page skin of the **Default** theme as seen in the Design view window in Visual Studio:



As with all the other skin files, the page skin includes a number of controls with predefined names. You can change the look of your community by simply changing the placement of these elements in the page skin.

For example, you can see that both ad spaces (the one at the top of the page, and the one at the right) are **AdRotator** controls. They are configured with data from the two XML files in the **Advertisements** folder as explained in Chapter 3. (You can identify the configuration file used to populate an **AdRotator** control by looking at its **AdvertisementFile** property.) Other examples are the **TextBox** control used for searching (named **txtSearch**) and the **Logo** control. Unlike the other controls, which are part of the .NET Framework, **Logo** is a CSK control. We'll learn more about some of the CSK controls later in the chapter.

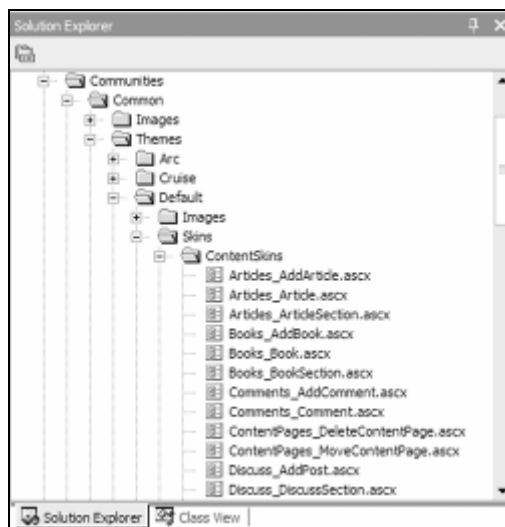
There are two placeholder controls. The most important one is **Content**, which is the main display area of the page. At run time, this is filled with content depending on the page that is being browsed by the visitor.

The content that fills the **Content** placeholder is generated by different content skins, depending on the section being browsed by the visitor. Content skins are skin files (that are also **.ascx** web controls) that fill the content area in the page skin.

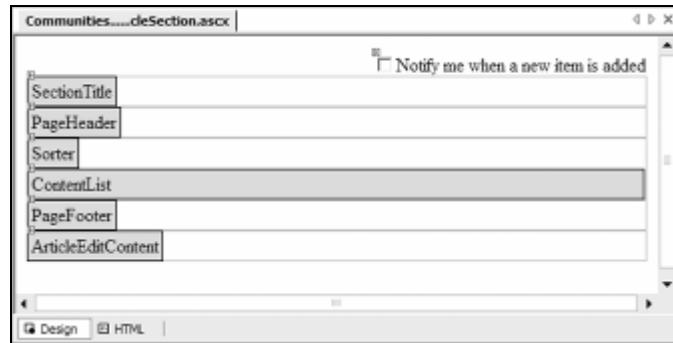
Content Skins

Content skins describe the layout of the content area of a community page. It includes pages such as the one that displays the list of articles or an individual article in the Mission section, or the page used to add a new article.

You can see the list of content skin files of the **Default** theme in the **Communities/Common/Themes/Default/Skins/ContentSkins** folder:



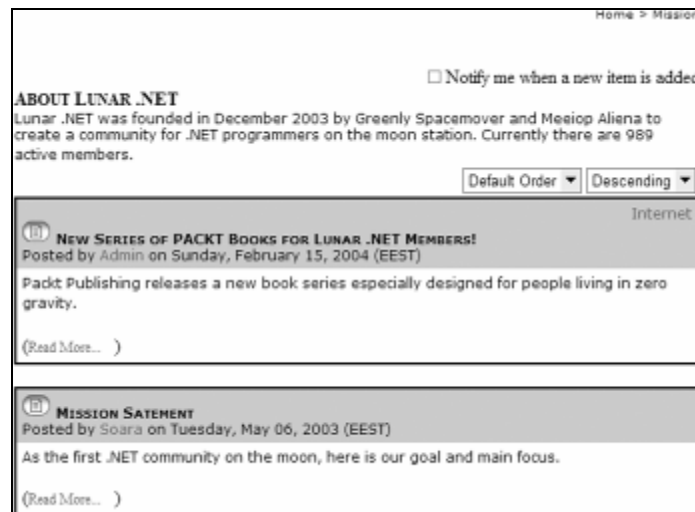
The file naming convention is quite intuitive, and it helps you guess what each file does (or to find the file you are looking for). When visiting the first page of a section (such as the Articles or Events page), a control with a name like `Articles_ArticleSection.ascx` or `Events_EventSection.ascx` is loaded. Controls suffixed by "Section" contain a `ContentList` control that displays the list of items in that section (such as a list of articles, events, books, and so on). The following screenshot shows the `Articles_ArticleSection.ascx` skin file. This is the control that generates the Articles (Mission) section in the Lunar .NET community.



Again, the real content of the skin is generated by the CSK user controls it includes. For now you just need to know that they generate the needed content, sometimes depending on the skin they are used in.

In `Articles_ArticleSection.ascx` file, perhaps the most interesting CSK control used is the `ContentList` control, which knows how to display lists of elements. For this page, it will display summary information about each article in the list (in the Lunar .NET community, go to the Mission section and you'll see).

The following screenshot shows `Articles_ArticleSection.ascx` in action:



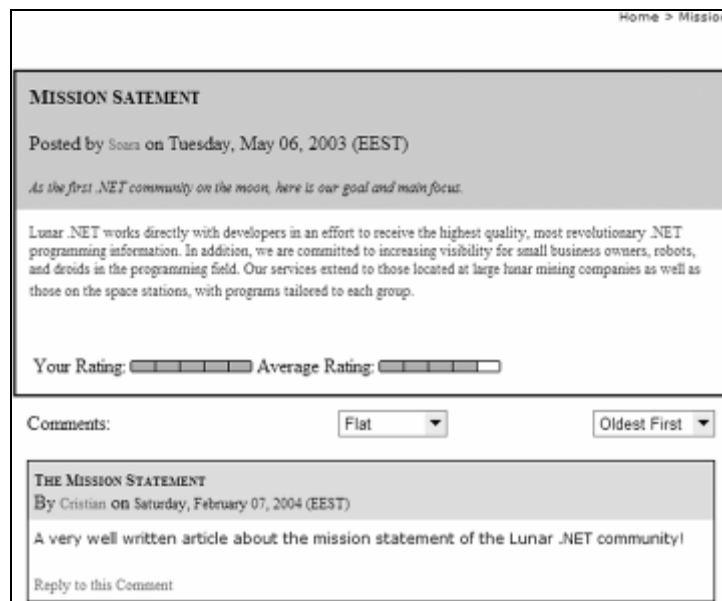
When viewing the details of a particular article (by clicking the [Read More](#) link), a details page opens. The content skin responsible for the details of a particular item in a section is generated by a control with a name like `Articles_Article.ascx`, or `Events_Event.ascx` (which are also content skins).

Take a look at the layout of `Articles_Article.ascx`, which describes how an individual article looks when displayed to the visitor:

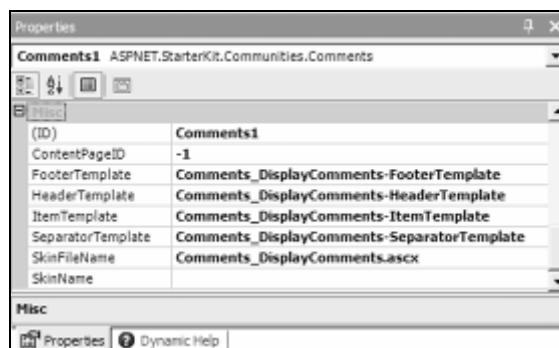


The CSK custom controls that form the skin have self-describing names (`Title`, `Author`, `Rating`, and so on), and most of them have a `CssClass` property, which references one of the styles from the current style (CSS file). It's easy to imagine how changing the CSS file affects the way the entire community website looks.

When rendered for a visitor by the CSK engine, the `Articles_Article.ascx` page looks like this:



Among the controls that are used in a content skin, some controls can have their own skins. For example, have a look at the properties of the **Comments** control, which displays the comments related to an article:

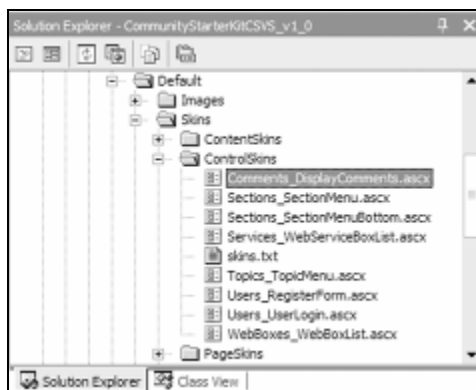


The **SkinFileName** property contains the name of another ASCX skin file. A skin that applies to a control inside the content skin is named a control skin.

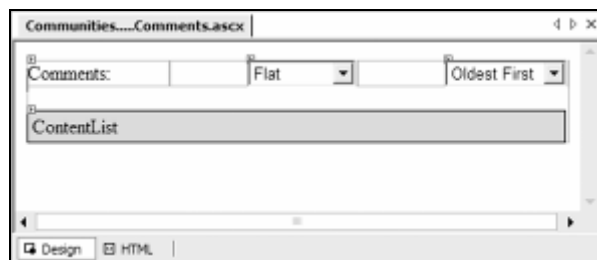
Control Skins

Control skins determine the layout of CSK controls. Since the look of most controls in a skin can be controlled exclusively through the CSS file, only a few controls are complex enough to use separate skins.

By taking a look in the **ControlSkins** folder of the **Default** theme, you can see the list of control skins:



Comments_DisplayComments.ascx is the control skin referenced from the **Comments** control and looks as in the following image, when viewed in the Design view window:

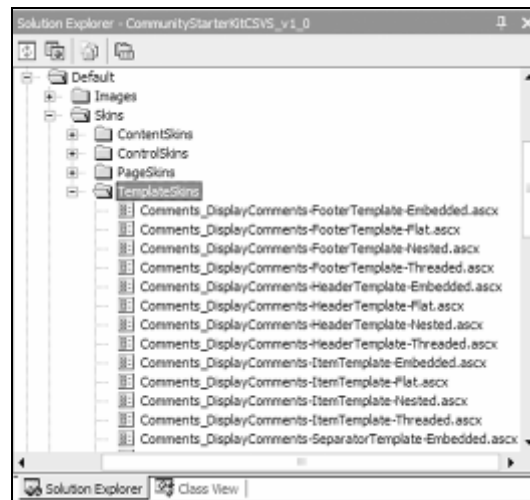


Template Skins

Template skins are used to determine the appearance of item lists inside a website section. They are used to generate the look of:

- Comments inside any section, including articles, events, and so on
- Comments in discussion forums
- Photo-gallery elements

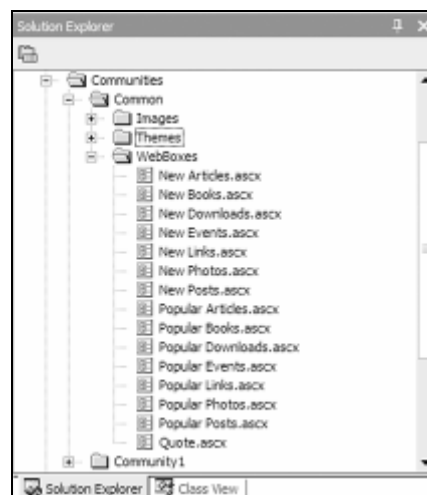
In the following screenshot, you can see some of the files in the **TemplateSkins** folder in the **Default** theme:



As we saw in Chapter 2, we have four display choices for comments: Flat, Nested, Threaded, and Embedded. For each display mode, there are four template files in the **TemplateSkins** folder: **FooterTemplate**, **HeaderTemplate**, **ItemTemplate**, and **SeparatorTemplate**. Depending on the display mode, a set of these files is used to populate a **DataList** control (a .NET control that displays lists of items).

The Web Boxes

WebBoxes is the third subfolder of the **Communities/Common/** folder. It contains skin files (also in the form of ASCX controls) that generate the Web Boxes. However, these skins apply to all themes. If you want to create a new local Web Box data source, you need to create a new skin file in the **WebBoxes** folder:



These files appear as information sources in the [Community Administration Menu | Edit Sections | Edit Section | Web Boxes](#). For a quick test, make a copy or rename one of the files in the [WebBoxes](#) folder, and then go to the Community Administration Menu—you'll see the change immediately.

Creating Themes

Creating a theme implies creating a new theme folder in [Communities/Common/Themes](#) for the theme to be visible from all communities, or in [Communities/community_folder_name/Themes](#) if the theme is to be visible to a single community.

The new theme folder must have the same structure as the already existing theme folders. In other words, it should have a [Styles](#) subfolder that contains one or more CSS files, and a [Skins](#) subfolder, with four subfolders: [ContentSkins](#), [ControlSkins](#), [PageSkins](#), and [TemplateSkins](#).

Once the new theme folder and its components are in place, the new theme (its skin and styles) will be visible in the [Edit Sections/Edit Section/Appearance](#) folder. You don't need to recompile or restart the application. The simplest way to test this is to copy one of the existing theme folders (such as [Arc](#) or [Frida](#)) to a folder named [MyCopiedTheme](#), in the same location.

No matter what kind of skin file you're developing (page skin, content skin, control skin, or template skin), the first step is to analyze one of the existing skin files from the [Default](#) theme, and see what CSK controls are used within. As an example, let's look at [Comments_Comment.ascx](#):



It's easy to see that the content is generated by the included CSK controls, such as [Title](#), [Author](#), [DateCreated](#), and so on. These controls are part of the [ASPNET.StarterKit.Communities](#) namespace. To get a better idea about how they are used, here's the code from [Comments_Comment.ascx](#):

```
<%@ Control %>
<%@ Register TagPrefix="Community" Namespace="ASPNET.StarterKit.Communities"
Assembly="ASPNET.StarterKit.Communities" %>
<table cellpadding="5" cellspacing="0" border="0" width="100%"
class="Comments_Table">
<tr>
<td class="Comments_IntroCell">
<community:Title id="lblTitle" CssClass="Comments_Title"
```

```

                                Runat="Server"/>
        <br>
        by
        <community:Author id="lnkAuthor" CssClass="Comments_AuthorLink"
                                Runat="Server"/>
        on
        <community:DateCreated id="lblDateCreated"
                                CssClass="Comments_DateCreated" Runat="Server" />
    </td>
</tr>
<tr>
    <td class="Comments_BodyCell">
        <community:CommentText CssClass="Comments_BriefDescription"
                                Runat="Server" />
        <p>
            <community:CommentReply Text="Reply to Comment" runat="Server" />
        </p>
        <community:CommentRating runat="Server" />
    </td>
</tr>
</table>
<p>
    <community:CommentParentLink Text="Return to Parent" id="lnkParent"
                                CssClass="Comments_ParentLink" Runat="Server"/>

```

Some of the CSK controls have a **CssClass** property, which reference styles defined in the CSS file being used.

The Community Starter Kit Custom Controls

We'll take a look at the common custom controls in the CSK.

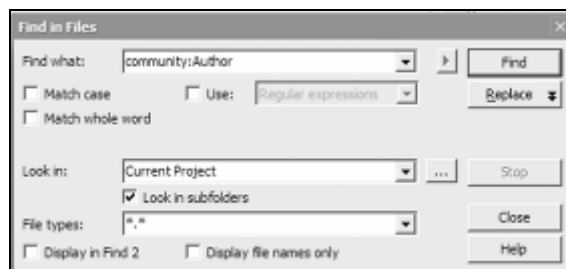
The secret of building a functional skin file lies in using the included CSK controls properly. You can find these custom controls under the **Controls** subfolders under the **/Engine** folder:

- **/Engine/Admin/Controls** subfolder contains the controls used in the admin pages.
- **/Engine/Modules/module_name/Controls** subfolder contains controls specific to particular modules. **module_name** can be one of the following: **Articles**, **Books**, **Discuss**, **Downloads**, **Events**, **Links**, **ParentSection**, or **PhotoGallery**. (There are two more folders that have no custom controls defined: **CustomModules** and **HtmlPage**).
- **/Engine/Framework/feature_name/Controls** subfolder contains various controls specific to CSK features. The subfolders in **Framework** that contain custom controls are: **Comments**, **ContentPages**, **Images**, **Moderation**, **Notify**, **Ratings**, **Controls**, **Services**, **Topics**, **Users**, **Voting**, and **WebBoxes**.

You can find a document that contains summary of details about all these controls at <http://www.communitystarterkit.com/Additional+Documentation/6052.aspx>. In Chapter 6 we'll learn more details about rest of the functionality you can find in **/Engine/Modules** and **/Engine/Framework**. For now, we'll focus on the common controls in **Engine/Framework/ContentPages/Controls**, which are the controls that appear in most CSK content skins.

Keep in mind that if you need to see the file in which a particular control is used, you can use the **Find in Files** feature. You can access this from **Edit | Find and Replace | Find in Files** (the default

shortcut is *Ctrl+Shift+F*). The **Find in Files** window in the following screenshot shows how you can search in the **Current Project** for the **Author** control:



The results are listed in the **View Results** window. The **Display file names only** checkbox will be useful when you only want to see the files that contain the specified control.

In case you need to find out where a particular class (including custom controls) is defined in the project, the Class view window in Visual Studio .NET will prove to be very helpful.

Learning about the Common Content Controls

The common controls include controls such as **Title**, **Author**, and **DateCreated** that can be used in any CSK skins. In many screenshots so far, you could see how skin files are created by including such controls. We will analyze the commonly used controls.

AddContent

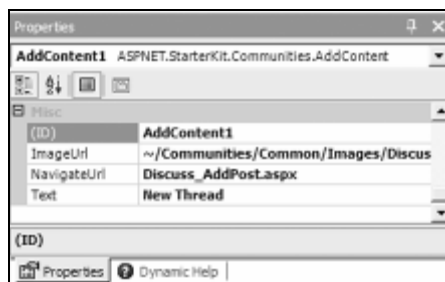
AddContent is a control used to display the **Add Item** links, optionally as images, in your content pages. Unlike the other controls, whose classes are saved in files with the same name, the **AddContent** class can be found in the **AddItem.cs** file. **AddContent** is used in a single skin file **Discuss_DiscussSection.ascx**:



In the preceding screenshot, you can see an **AddContent** control (the upper one) and a **DiscussEditContent** (the lower one) control. **DiscussEditContent** is a control specific to the Discuss module, and it is included in the page footer (take a look at the HTML code in **Discuss_DiscussSection.ascx** and you'll see).

Perhaps not very intuitively, in most cases the **Add Item** links (such as **Add Article** or **Add Book**) are generated with module-specific, edit content-type controls (such as **ArticleEditContent**, **BookEditContent**, and so on), which are capable of showing **Add**, **Edit**, **Delete**, **Move**, **Comment**, and **Moderate** links, but are unable to show images instead of text links, like the generic **AddContent** control.

Discuss_DiscussSection.ascx is the only skin file in the CSK to use the **AddContent** control. From the **Properties** window for the **AddContent** control, you can view its specific properties:



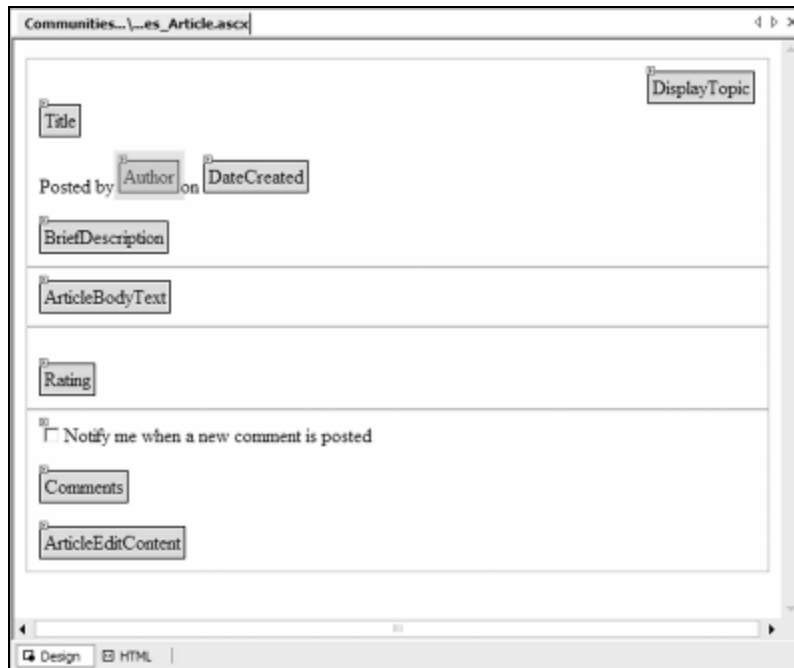
The properties are:

- **ImageUrl**: This is an optional property that contains the path to the **Add Item** picture.
- **NavigateUri**: This indicates the **Add Item** skin page.

- **Text**: This is the text that gets displayed if **ImageUrl** is not specified. It is also used to generate the **Alt** attribute in the **img** tag when an image is displayed.

Author

The **Author** control displays the author of the current content page. Open the **Articles_Article.ascx** skin in Visual Studio to see how it appears in a skin file:



Switch to Internet Explorer (or your favorite web browser) and click on the **Mission** link of the Lunar .NET community (which gets you to the Articles page of the community).

You can also see an author link on the **Articles_ArticleSection** page (the one that appears immediately after clicking the **Mission** link) before clicking on the **Read More** link that forwards you to the **Articles_Article** page. However, that link is generated by the **ContentList** control, which displays lists of elements. The **ContentList** control uses internally the **ItemAuthor** control to display the **Author** link. Please see the **ContentList** and **ItemAuthor** controls for more details.

Next click on the **Read More** link for one of the articles to see the **Author** control in action:



If the author is a member of the local community, a link to his or her profile page is displayed. Otherwise, the image specified by the `ExternalImage` property is displayed; the default value of this property in the control is `~/communities/common/images/external.gif`.

You can apply a `CssClass` style to your `Author` control, to be used when displaying author links. The `Author` control in `Articles_Article.ascx` page has its `CssClass` property set to `Article_AuthorLink` (the default `CssClass` property defined in the control is `author`).

BriefDescription

The `BriefDescription` control displays a brief description of the content page in which it is included. It is also used in `Add Content` pages with preview functionality.

The brief description that appears in lists displayed with the `ContentList` control, when displaying lists of items (as in `Articles_ArticleSection`), is generated by the `ItemBriefDescription` control.

A content page can be set to accept HTML for the brief description from `Edit Sections/Edit Section/Advanced`. In each case, the `BriefDescription` control correctly displays the output depending on the `Allow HTML` value.

When using `BriefDescription` in your own controls, you'll want to set its `CssClass` property. The `CssClass` property for the `BriefDescription` in the `Articles` page is `Article_BriefDescription`.

CommunityImageButton

The `CommunityImageButton` control is used to generate image button controls and works the same way as the ASP.NET `ImageButton` control. The only place where you can see a `CommunityImageButton` in the CSK is the page skin (`Default.ascx`) of the Cruise theme, where it displays the `Go` button below the `Search` text box.

In code, the `CommunityImageButton` control differs from the ASP.NET `ImageButton` control, since it automatically transforms the `ImageUrl` property containing the path to the image from a relative path to an absolute path. This ensures that the button image appears correctly in the CSK pages.

Here's the code for the control in C#. You can see that it inherits all functionality from `ImageButton`, and on initialization it changes the `ImageUrl` property:

```
[Designer(typeof(ASPNET.StarterKit.Communities.CommunityDesigner))]
public class CommunityImageButton : ImageButton
{
    override protected void OnInit(EventArgs e) {
        base.OnInit(e);

        // set the ImageUrl to app relative path
        ImageUrl = ResolveUrl(ImageUrl);
    }
}
```

ContentList

This is one of the more complex controls in the CSK (a list of skin files that use the `ContentList` controls is given in Appendix A). It displays lists of items such as lists of books or articles. Feel free to open a few of them in Visual Studio to get an idea about what kind of content the `ContentList` generates.

The `ContentList` control is also used in all the Web Boxes web controls (`NewArticles.ascx`, `NewBooks.ascx`, and so on), except `quote.ascx`.

The `ContentList` control is similar to the `DataList` and `DataGrid` controls in the following ways:

- It can be bound to data sources.
- It supports templates (header, footer, item, alternating item, separator, and no content).
- It can display data in multiple columns.
- It supports paging.

To see the properties of the `ContentList` control without dissecting its code (`ContentList.cs`), you can open a skin file that uses it in Design view, selecting the `ContentSkin` object and looking at the `Properties` window.

Both Visual Studio .NET 2002 and Visual Studio .NET 2003 interfere with the `<community:ContentList>` section in the ASCX skin file (by removing its template sections) when you change properties of the `ContentList` object using the **Properties** window. If you want to test this, make a copy of the skin file first! You'll need to make the necessary changes to the `ContentList` object working directly in HTML view. You can still use the Design view to edit the other controls in the page. Also, you may want to use the Design view to view the properties of the `ContentsList` object, without modifying them.

Keep this in mind if you want to play with the control's properties while reading the properties list that we'll see shortly.

The basic structure for defining a `ContentList` object is typically like this:

```
<community:ContentList id="ContentList"
                        PagerText="Page {0} of {1}" Width="100%"
                        Runat="Server">
...
...
</community:ContentList>
```

The `ContentList` object supports a number of properties that define its look and functionality. For now, you can see in the HTML code how its property values are being set to change their default values. Usually, the properties that are left with their default values aren't also assigned explicitly.

This is also the default way Visual Studio .NET works in Design view mode. When you add and configure controls in Design view, Visual Studio only generates code to set the properties that have non-default values.

Note that you don't need to recompile the project when modifying control property values in skin files. You only need to recompile the project when you modify C# source code.

The following are the most important properties of `ContentList`:

- **CurrentPage**: Gets or sets the current page of contents.
- **DataSource**: An `ArrayList` object used to populate the list.
- **RepeatLayout**: Can be **Flow** or **Table**. When set to **Table**, the content items are displayed using **TABLE** instead of **SPAN** HTML tags.
- **RepeatColumns**: The number of table columns to generate; it works only when **RepeatLayout** property is set to **Table**.
- **PagerText**: The text that displays the current page and the total number of pages. It doesn't have a default value in the control, and the typical value is **Page {0} of {1}**. The pager text is displayed only when there is more than one page of content items.
- **TotalRecords**: Used for the total number of items that are displayed by the control. You don't need to deal with this property directly. It is used internally to calculate the number of content pages, by dividing **TotalRecords** property by the number of records per page. The default number of records per page is set by default at 10 in

`Engine/Framework/Sections/Components/SectionInfo.cs` (the `recordsPerPage` class member). You'll need to edit this file and recompile the project if you want to change the number of records per page.

Some of these properties, such as `CurrentPage`, `DataSource`, or `TotalRecords`, aren't to be used directly in the skin files (it wouldn't make much sense to establish the current page at design time). Instead, they can be accessed from code-behind files that instantiate `ContentList` objects, or used internally by the `ContentList` control.

When using a `ContentList` control, apart from setting its properties, another important step is to implement its templates. Remember that `ContentList` works like `DataList` or `DataGrid` controls (without templates, it doesn't know what to display).

Unlike the `DataList` or `DataGrid` controls, the `ContentList` control doesn't allow you to edit its templates directly in the Visual Studio .NET Design view. You need to edit them manually in HTML view.

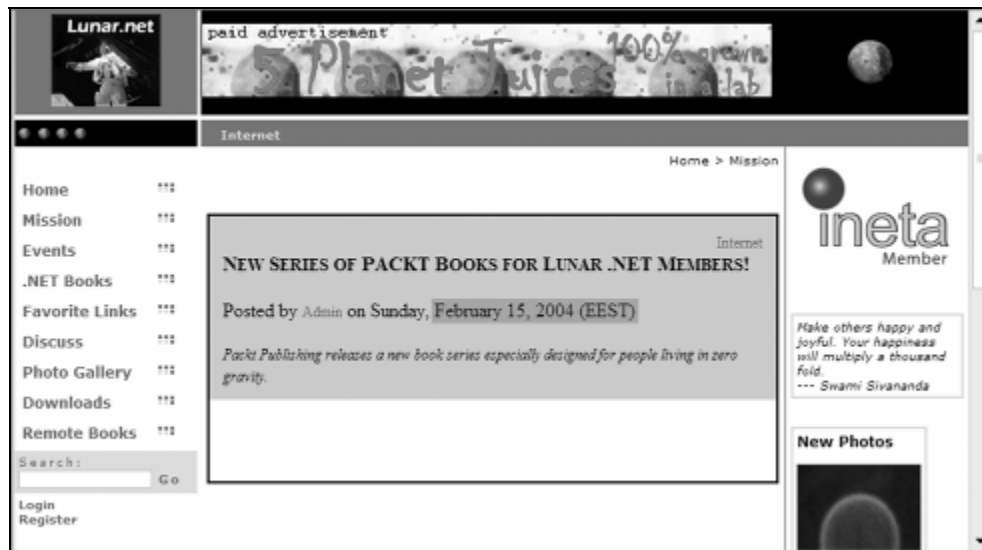
There are six template types supported by the `ContentList` control. Let's see a short description of each of them before moving to the other details:

- **HeaderTemplate**: Contains the code to be generated at the top of all the content items.
- **ItemTemplate**: Contains the code that is generated for each item displayed by the `ContentList` control (remember that `ContentList` is used to display lists of elements!).
- **AlternatingItemTemplate**: When set, it is used to generate alternating HTML code after each `ItemTemplate` generated content item.
- **SeparatorTemplate**: Generates HTML that appears between two content items.
- **FooterTemplate**: Contains the code to be generated at the bottom of all content items.
- **NoContentTemplate**: Contains the message to be generated when no content items exist.

In the `ContentList` template sections, you'll usually see HTML code that also contains CSK custom controls. The CSK engine contains a number of controls specially created to be used within `ContentList` templates; their names are prefixed by `Item`, and their functionality simulates the functionality of other existing CSK controls (for example, `ItemAuthor` and `ItemBriefDescription`). These are similar to `Author` and `BriefDescription` controls.

DateCreated

The `DateCreated` control displays the date on which a content item was posted to the community website. The `ContentList`-version of this control is named `ItemDateCreated`. The following screenshot shows it in action in `Articles_Article.ascx` page:



The two properties you need to know about are:

- **DateFormatString** contains date formatting information and its default value is "{0:D}", which displays the date as a long date. Look at the .NET Framework documentation for the **DateTimeFormatInfo** class for a complete list of date format characters.
- **DisplayTimeZone** is a boolean property that specifies if the time zone should be displayed or not.

DisplayQuota

DisplayQuota control is used to prevent users from uploading content over the established community or user quota, and to display statistics about the current quota settings and usage. You can find **DisplayQuota** in:

- **AddImage.aspx**
- **EditSection.aspx**
- **AddTopic.aspx**

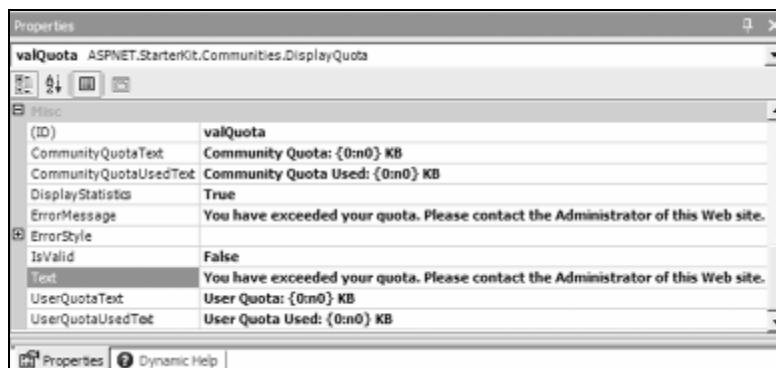
For the list of skin files, please read Appendix A. As the **EditSection.aspx** control (the form that generates all **Edit Section** pages in the Community Administration Menu) includes a **DisplayQuota** control, you effectively can't do any administrative tasks with that page until you fix the quota problem. Even if the **Apply Changes** buttons appear to work, they don't! In the **EditSection.aspx** control, the **DisplayQuota** control is visible only if the quota was exceeded.

The only sections where users can add binary content are **Downloads** and **Photo Gallery**. Thus they are the only ones that include a **DisplayQuota** control. The following screenshot shows how the quota statistics are displayed on the addition of a new photo by the administrator:



Regular users are only shown their user quota. Only administrators are also shown community quota information. **DisplayQuota** control has a number of properties that set the way quota statistics or error messages are displayed.

CommunityQuotaText, **CommunityQuotaUsedText**, **UserQuotaText**, and **UserQuotaUsedText** are self-explanatory since they display the "Community quota" type of text. The following screenshot shows their default values:



A boolean property named **DisplayStatistics** establishes whether the **Statistics** box is visible or not. You can alternatively set the **Visible** property to **False**, but this prevents the error message from being displayed (see the **RenderErrorMessage** method), and it doesn't prevent the control from doing some initialization procedures that it wouldn't normally do when the **DisplayStatistics** property is set to **False** (see the **OnInit** method).

The properties related to displaying quota error messages are **Text**, **ErrorMessage**, and **ErrorStyle**. The **Text** property defines what gets displayed when exceeding the community quota and **ErrorStyle** determines how the message will look.



The default **CssClass** value for the **DisplayQuota** property is **displayQuota**.

The maximum length of the file being uploaded is also limited by the value of the **maxRequestLength** attribute of the **httpRuntime** element in the **Web.Config** file. It specifies the maximum upload size in kilobytes. The default value in the CSK is **10000** (overriding the ASP.NET default of **4000**). You may want to add the **executionTimeout** attribute to extend the maximum time that a request is allowed to execute (the default is 90 seconds).

HtmlTextBox

This control generates the HTML editor used to edit or add new content to the community website. We also discussed it in Chapter 2.

HtmlTextBox is also used in the admin pages in **SectionAppearance.ascx** (located in **/Admin/EditSections**) to edit the Page Head and Section Page Footer for the section that's being edited.

The properties of the **HtmlTextBox** control are as follows:

- **AllowHtml**: Specifies if the editor should allow HTML content. Possible values are **None**, **Limited**, and **Full**, which are discussed in Chapter 2. With other words, the value for **AllowHtml** is inherited from the website section in which the **HtmlTextBox** control is placed.
- **Rows**, **Columns**: Defines the size of the control; the default value for **Rows** is 13 and for **Columns** is 50.
- **IsUpLevel**: Is **True** if the client browser is Internet Explorer 5.5 or higher. Its value is automatically detected in code (see the **DetermineUpLevel** method). For other browsers and older versions of Internet Explorer, the control will display a simple textbox instead of the HTML Editor.
- **Text**: Represents the contents of the editing box.

ItemAuthor

Similar to the **Author** control, except it must be used from within **ContentList** controls.

ItemBriefDescription

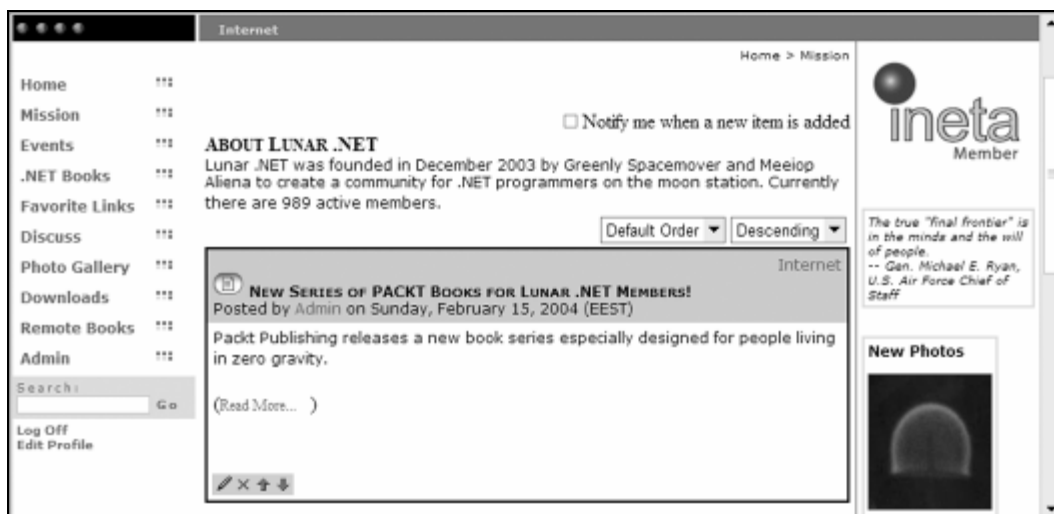
Similar to the **BriefDescription** control, except it *must* be used from within **ContentList** controls.

ItemDateCreated

Similar to the **DateCreated** control, except it must be used from within **ContentList** controls.

ItemEditContent

This control is used to generate edit, delete, and up/down links, for users with sufficient permissions. It can be used only within **ContentList** objects. The following screenshot shows how this control looks like inside the **Mission** section of the Lunar .NET community:



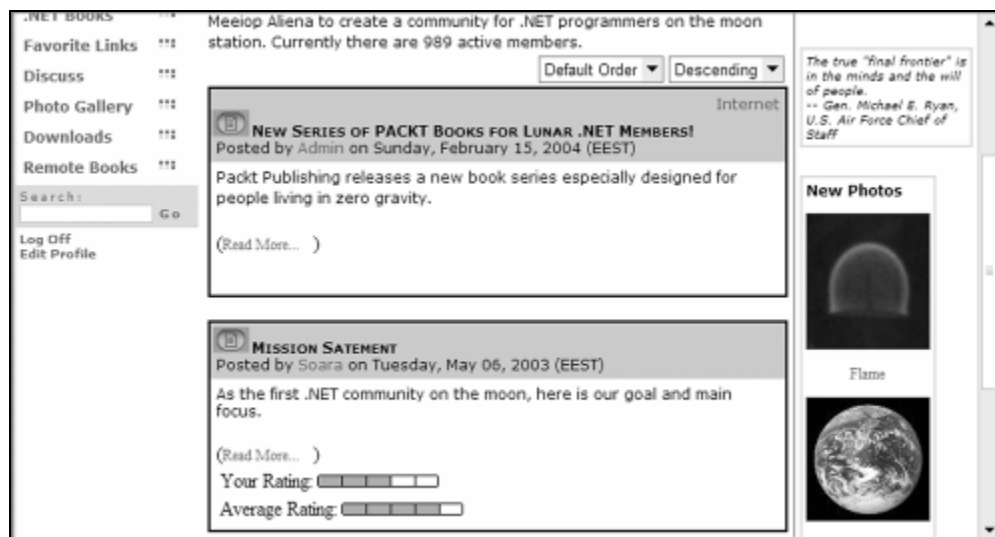
ItemHasRead

ItemHasRead control must be used from **ContentList** controls. It displays images that show if the logged-in user has viewed the current content item.

Its properties are:

- **ReadImage**: Path to the image for already viewed items. The default value, declared inside the control, is `~/Communities/Common/Images/HasRead/Read.Gif`.
- **NotReadImage**: Path to the image for items that haven't been viewed yet. The default value is `~/Communities/Common/Images/HasRead/NotRead.Gif`.

The only skin file where `ItemHasRead` is used is `Articles_ArticleSection.ascx`. `Read.Gif` and `NotRead.Gif` differ only in color, and so you might not see the difference in the following grayscale picture:



ItemImage

`ItemImage` is an abstract control class (inherits from `Control`) that generates an image with a specific width and height. These are used to display images retrieved from the database.

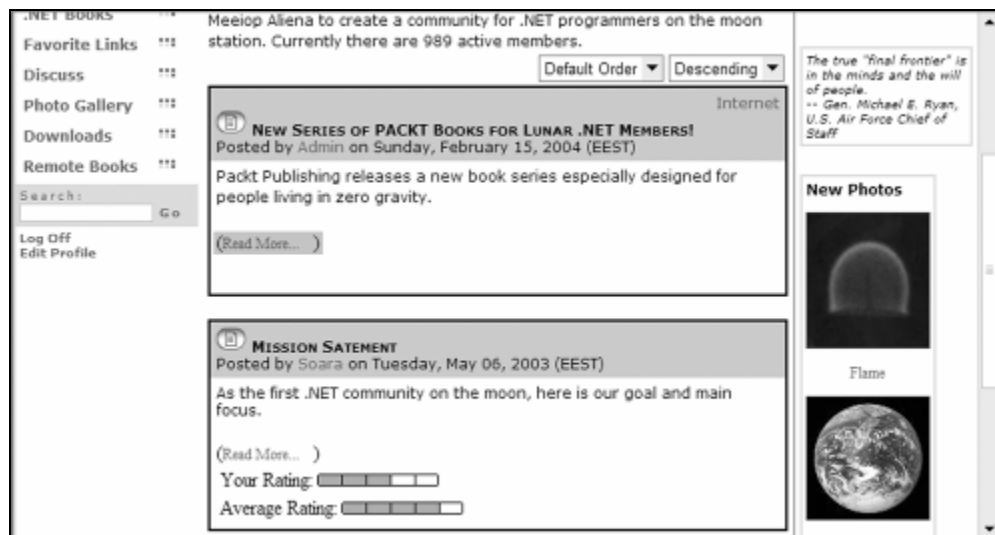
`ItemImage` is an abstract class, so it cannot be instantiated directly. It is inherited by `ItemBookImage`, `ItemPhotoImage`, and `ItemEventImage`, which are module-specific controls. These three classes must be used from within `ContentList` controls (and are called the `ItemXXX` controls).

The properties of the `ItemImage` class are self-describing: `Height`, `Width`, and `Border` are of type `int` and `Align CssClass`, `ImageUrl`, and `NavigateUrl` are of type `String`.

ItemReadMoreLink

The `ItemReadMoreLink` control generates the `Read More` links inside `ContentList` templates.

The `Text` property (whose value is by default `Read More`) can be used to change the text output of the control. The following screenshot shows `ItemReadMoreLink` in `Articles_ArticleSection.ascx`:

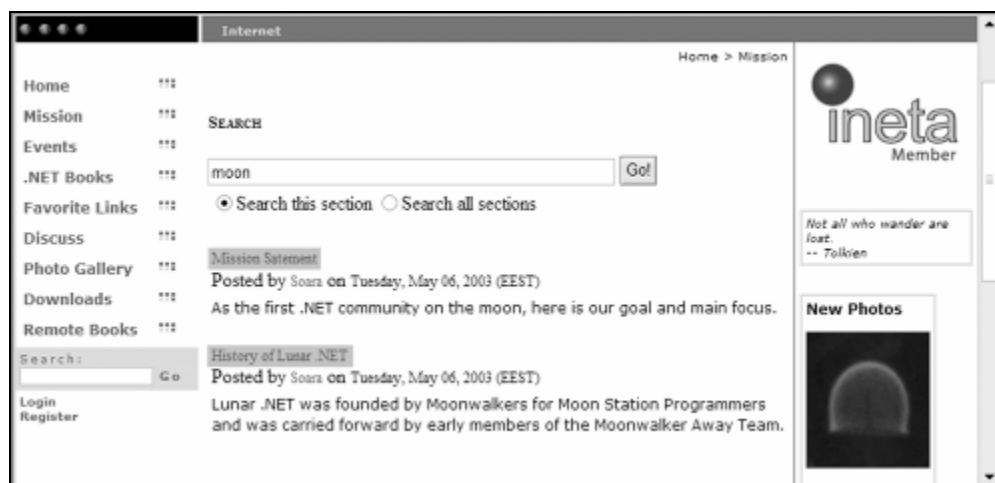


ItemTitle

The `ItemTitle` control is similar to the `Title` control (which is discussed later), except it must be used from within `ContentList` templates.

ItemTitleLink

The `ItemTitleLink` control displays an item title as a link, in a `ContentList` template. It is also used in the Web Boxes ASCX controls (except `quote.ascx`), which you can find in the `Communities/Common/WebBoxes` folder. Here you can see the links on the [Search](#) page:



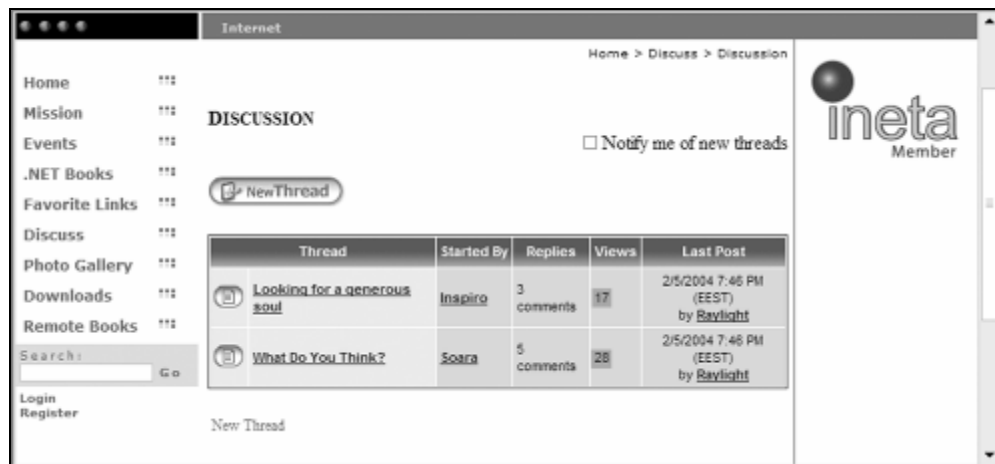
ItemViewCount

The **ItemViewCount** control displays the number of times a content item has been accessed, in a **ContentList** template. The control simply outputs a numeric value.

The only skin file in which this control is used is **Discuss_DiscussSection.ascx**. It's very easy to implement it in other sections. Just copy the following line in the **ItemTemplate** section of the **ContentList** in another skin file (such as **Articles_ArticleSection.ascx**), and you'll see that it works without you needing to write a single line of code:

```
<community:ItemViewCount runat="Server" ID="Itemviewcount1"/>
```

ItemViewCount displays the number of views for each message. In the following screenshot, you can see the **ItemViewCount** control in **Discuss_DiscussSection.ascx**:



LengthValidator

LengthValidator is a validator control (derived from **CustomValidator**) that checks the length of user input. If the length is greater than the allowed maximum value, the form can't be submitted.

LengthValidator is also used in the Admin part of the site, in two edit section controls: **SectionAppearance.ascx** and **SectionGeneral.ascx**. These pages were described in Chapter 3.

The following screenshot shows **Articles_AddArticle.ascx** in Design view. You can see the **LengthValidator** control highlighted.

When using the `LengthValidator` control, you need to set its `ControlToValidate` property, which sets the control being validated (usually a `TextBox` or `HtmlTextBox` control), and the `MaxLength` property.

Logo

The `Logo` control generates the community logo image that acts a link to the default page. If a section doesn't have a community logo set, it'll inherit the logo image from the parent section. By default, in the Lunar .NET community, only the home page has a community logo set and the others inherit the option.

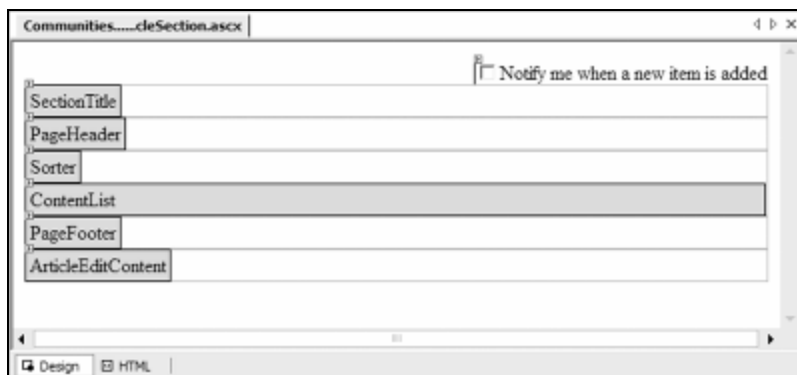
PageFooter

The `PageFooter` control displays the footer in the content area of a section. The community administrator can set the page footer by editing it in a `HtmlTextBox` control, in the [Community Administration Menu | Edit Sections | Edit Section | Appearance](#) menu.

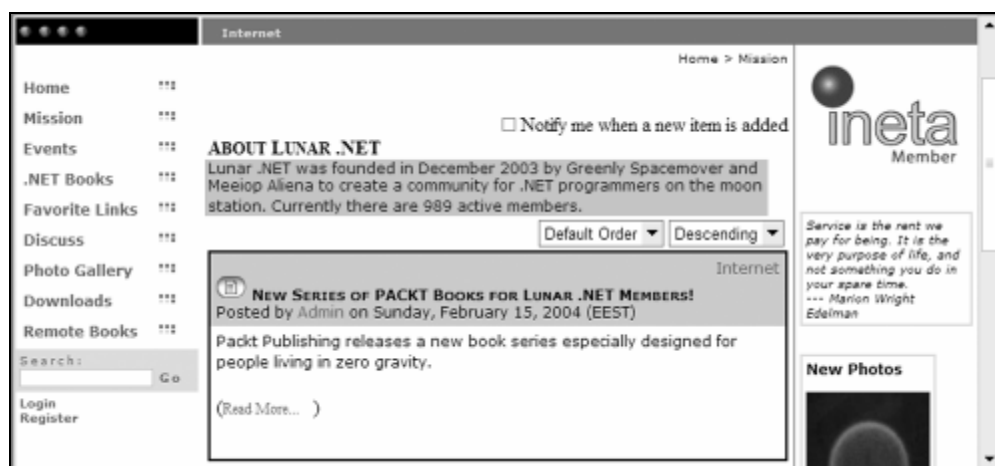
PageHeader

The `PageHeader` control is similar to the `PageFooter` control, except it renders the header of a community content page. `PageHeader` is used in the same skin files as `PageFooter`.

`Articles_ArticleSection.aspx` is one of the skin files that implement a page header:



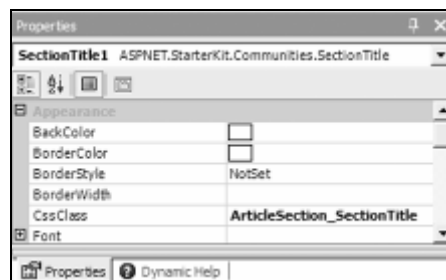
When rendered for the visitor, it looks like this:



The text of **PageHeader** displays can be set from [Community Administration Menu | Edit Sections | Edit Section | Appearance | Section Page Header](#). Note this text is stored differently than [Community Administration Menu | Edit Sections | Edit Section | General | Section Description](#), which represents the description that appears in the parent section.

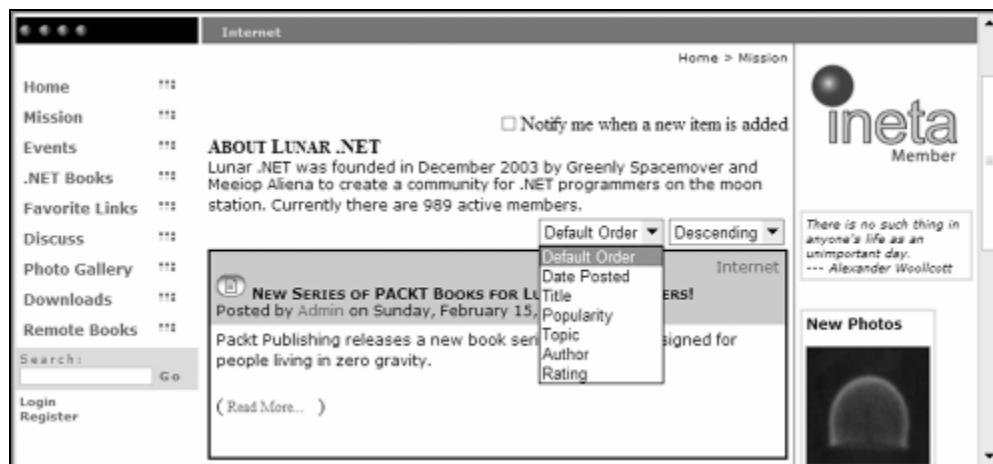
SectionTitle

The **SectionTitle** control displays the title of a content section. The default **CssClass** property defined in the control is **sectionTitle**. However, this value is overridden in most skin files where the **SectionTitle** control is used. In **Articles_ArticleSection.ascx**, its value is **ArticleSection_SectionTitle**:



Sorter

The **Sorter** control allows the visitor to sort lists of community content items displayed with the **ContentList** control. The following screenshot shows the control in action in `Articles_ArticleSection.ascx`:



The **Sorter** contains two dropdown lists. The first establishes the criterion on which to sort and the second establishes the sort order. The sort-order criterion can be one of the following: **Date Posted**, **View Count**, **Rating**, **Title**, **Date Commented**, **Date Updated**, **Author**, **Default**, and **Topic**. However, only the criteria that apply to the section being displayed are shown. For example, **Rating** and **Topic** aren't displayed if these options are disabled for the section.

The properties of **Sorter** are:

- **AscendingText**: The text that gets displayed in the second dropdown list for the ascending sort order; the default value coded in the control is **Ascending**.
- **DescendingText**: The text that gets displayed in the second dropdown list for the descending sort order; the default value coded in the control is **Descending**.

- **FlipSortOrder**: A boolean value that establishes the order in which **AscendingText** and **DescendingText** get displayed in the second dropdown list; the default value is **False**.
- **SelectedSortOrder**: This contains the currently selected sort order, containing the values from both dropdown lists. It is a read-only property automatically generated by the control.
- **SortColumnValue**: A string value containing the column used for sorting; the typical value you set is **Default**.
- **SortOrderValue**: A string value containing the sort order; it can be either **desc** for descending or **asc** for ascending order.

Title

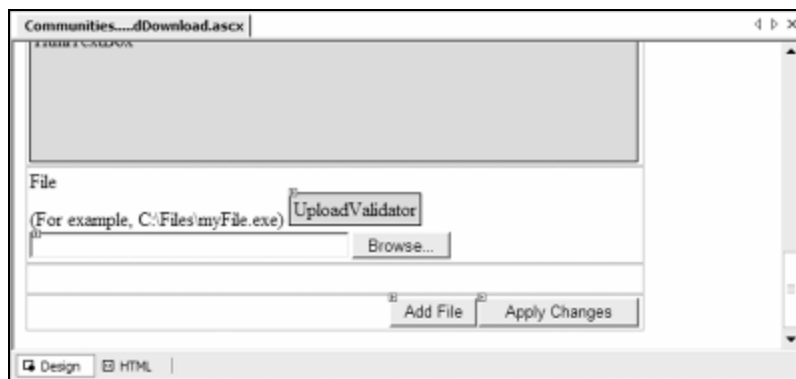
Title displays the title of a content item. Please refer to Appendix A for a list of the skin files.

UploadValidator

UploadValidator verifies if any uploaded files were transferred correctly to the community. It checks if the uploaded content size is greater than zero. **UploadValidator** is used in the following skin files:

- **Downloads_AddDownload.ascx**
- **PhotoGallery_AddPhoto.ascx**

The following screenshot shows the **UploadValidator** control in **Downloads_AddDownload.ascx**:



Here are its properties:

- **ControlToValidate**: The name of the **input type="file"** control to validate (for example, **txtDownloadFile** in **Downloads_AddDownload.ascx**).
- **ErrorMessage**: The message to display when the file wasn't uploaded properly.

ViewCount

ViewCount displays the number of times a content item has been accessed. It is intended to be used directly in skin files such as **Articles_Article.ascx**.

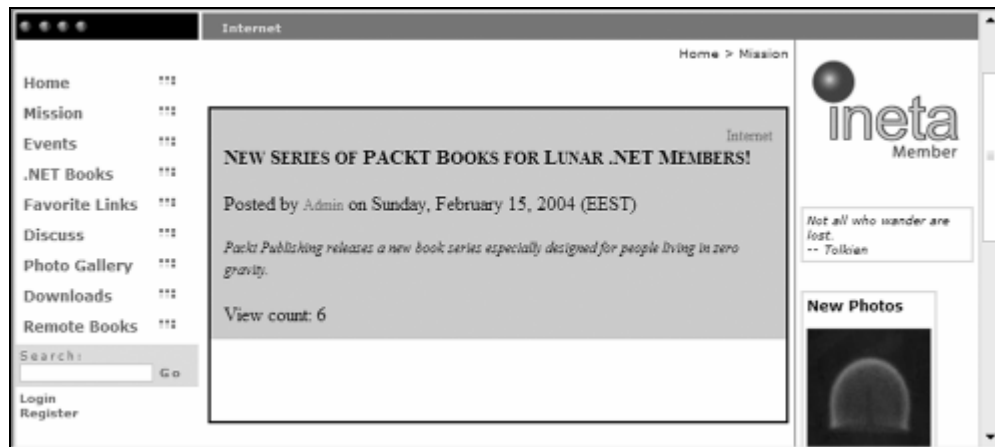
Open **Articles_Article.ascx** in HTML view, and add the following code just below the **BriefDescription** control:

```
<P>View count:
    <community:ViewCount runat="server" id="MyViewCount"
/>
</P>
```

Then switch to Design view. It should look like this:



The control will take care of everything now. You don't need to write a single line of code! After saving the page, reload your CSK website, and open one of the article pages:



Summary

A theme is composed of skins and styles. In this chapter we looked at what themes, skins, and styles are. You learned that creating skins is not such a complicated task if you leverage the functionality built-in CSK custom controls. By combining a skin file with a style, you create a brand new theme to make your community stand out from the others.

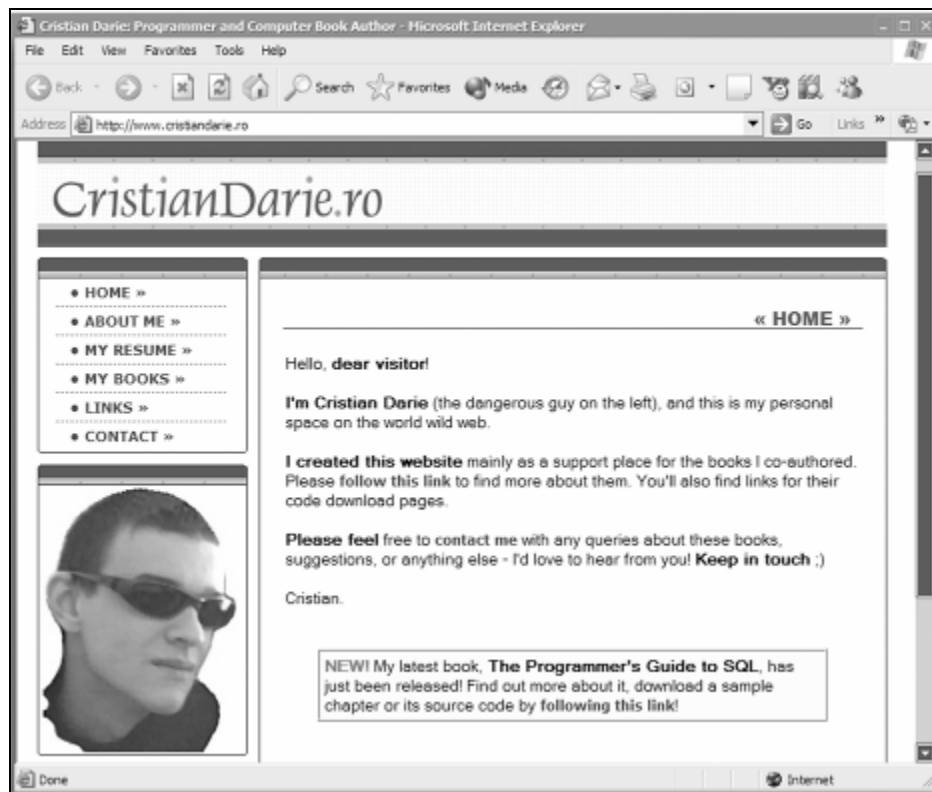
The next chapter will look creating a new community website from scratch.

5

Customizing your Website

Packed with the knowledge from the previous chapters, we're finally ready to build a complete and functional community website from scratch with a personalized look and feel.

Let's first establish what functionality we want for the new website and how it should look. To make things easier and reuse the layout of an already existing website, we'll create my personal community website. We aim to have it look like my current website, <http://www.CristianDarie.ro>:



Building this website with the CSK will make use of all the colorful but extremely powerful features of the CSK. The website we're building needs to have the following features:

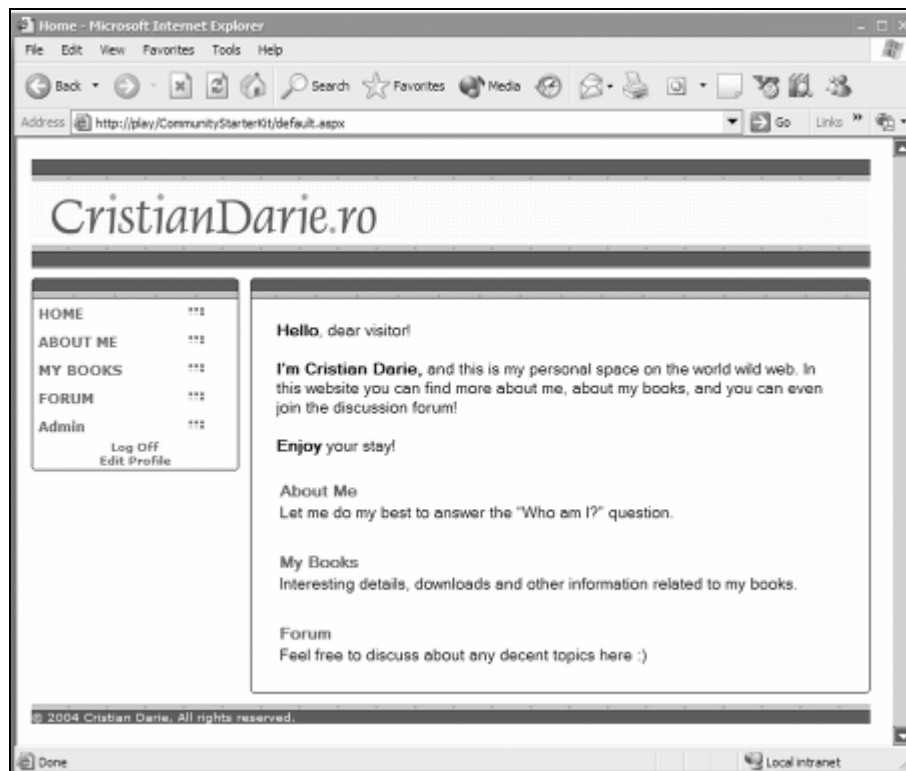
- Home
- About Me
- My Books
- Forum

We make use of CSK's security features to allow only authenticated users to post messages on the forum or to add comments in the My Books section. This also opens the possibility of sending newsletters to people that ask to receive them.

In case you were wondering, CSK doesn't have support for multiple languages. However, you can change the language of the website by modifying the skin files. More information on creating multi-lingual sites can be found in these forum topics on the <http://www.asp.net/> forums:

- <http://www.asp.net/Forums/ShowPost.aspx?tabindex=1&PostID=166265>
- <http://www.asp.net/Forums/ShowPost.aspx?tabindex=1&PostID=464438>
- <http://www.asp.net/Forums/ShowPost.aspx?tabindex=1&PostID=173535>

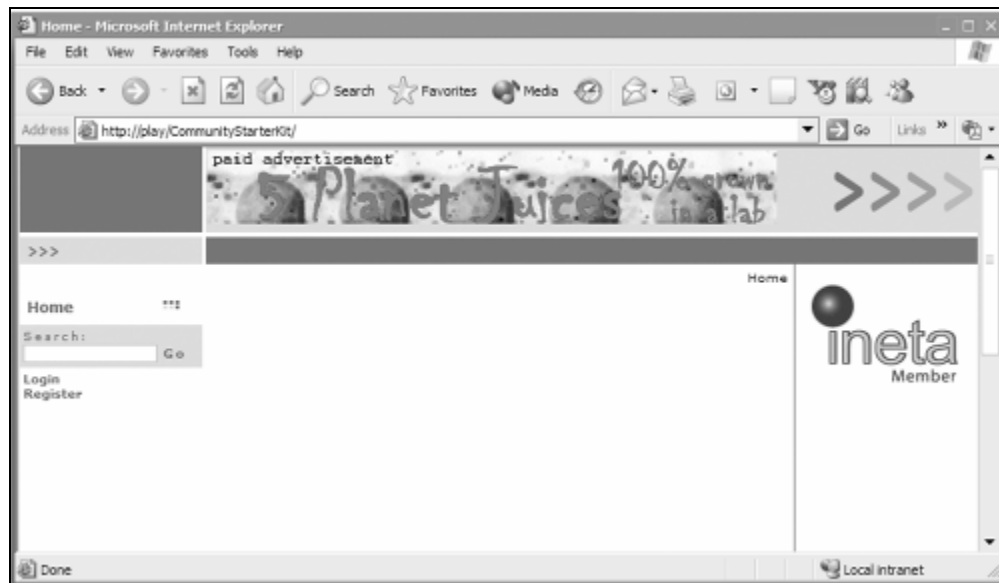
The new website, which takes a couple of hours to create (with the knowledge from the previous chapters and the HTML layout ready), will look something like this:



Throughout the chapter there are references to technical details such as database table names, stored procedures, C# methods, and so on. You'll learn more about these in the following chapters.

Setting up a New Community

Following the steps from the *Creating a New Community* section at the end of Chapter 3, create a new community on your machine named **Play**. When creating the new community in the ISP Administration page, uncheck the Apply Sample Content checkbox, as we want to start with a brand-new community site. Also, you may prefer to have a personalized admin account instead of Admin. Before moving on, make sure that you're shown a bare community website when accessing <http://play/CommunityStarterKit/>:



The rest of the chapter is structured in two parts. In the first part, we'll add the information to the website, and in the second, we'll change its layout. You may prefer to combine these two, but for the purposes of this chapter, it's easier and *cleaner* to deal with things one by one.

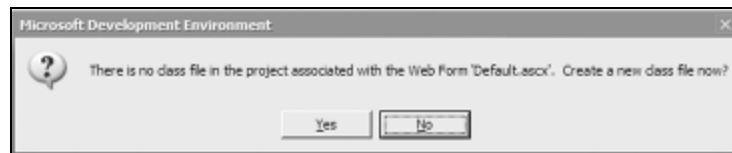
Preparing the Folder Structure and the Page Skin

Even though we'll deal with creating the new theme later, we'll prepare its folder structure by following these steps:

10. In `Communities/Common/Themes`, create the structure for a new theme called Blue:



11. In Styles, copy the `Default.css` file from the `Default` theme. Use the same method to copy the `Default.ascx` page skin from the `Default` theme folder to the `Blue` theme folder. Later, we'll modify this skin to create our own.
12. You'll be asked if you want a new class file created for the control. Choose No:



13. After copying the two files, your `Blue` folder should appear as follows:



That's it! The `Blue` folder will be the playground later when implementing the personalized look for our website. First, we'll add some substance to the website by adding a number of new sections.

Adding Information

We'll populate the website as follows:

1. Create a new Home section.
2. Add the About Me page.
3. Deal with the My Books section.
4. Add the discussion forum.

Start your engines!

Having a New Home

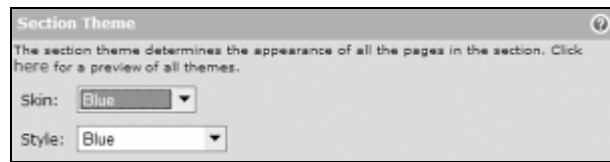
By default, when creating a new community, the Home section is an HTML Page section without any functionality and without the possibility of changing its appearance. However, if you choose **Apply Sample Content** when creating the community, the home section is automatically configured as a Parent section.

We will change the Home section to a Parent section. Among other features, this means that we'll get the useful listing of Child sections. Let's do this as follows:

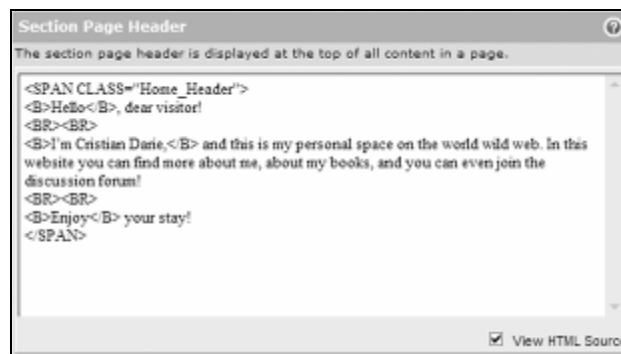
1. Log in as Admin, and go to the Community Administration Menu. Here you need to delete the Home section, but this is not possible since currently it is the only section. For now, edit the Home section to change its Section Name to `OldHome`. This will allow us to create a new section named Home. Click **Apply Changes**.
2. In **Edit Sections**, click on **Add New Section**, and set the following properties in the **General** tab:

Name	Value
Section Name	Home
Section Title	Home
Section Menu Title	HOME
Section Description	Welcome!
Section Type	Parent Section

3. Switch to the **Appearance** tab. The first detail we modify here is the theme. Choose the `Blue` skin and the `Blue` style (which refers to the `Default.css` file located in the `Blue` folder):



4. Next, let's edit the Section Page Header. Because we want to edit it in HTML mode, we click the View HTML Source checkbox and type the following HTML:

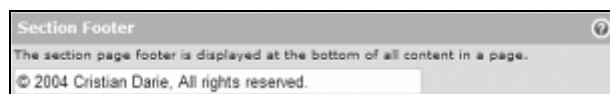


Note that we make use of a style (Home_Header) that doesn't exist in the current Default.css file. Later, when building the layout, we'll create the new styles and modify some of the existing ones.

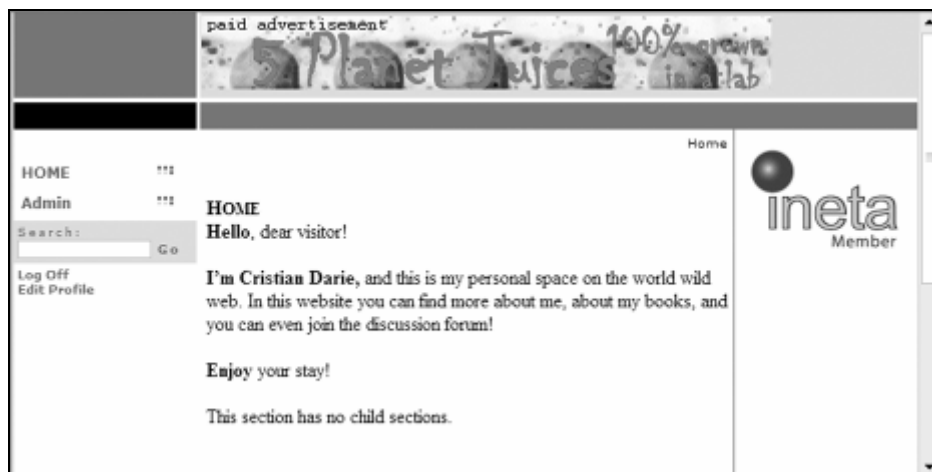
5. In the Appearance tab, you can also set the meta-description and keywords, helping the website to rank higher in search engine results. Here's an example:

Name	Value
Section Meta Description	Cristian Darie's personal home page
Section Meta Keywords	ASP.NET, Community Starter Kit, CSK, C#, programming, SQL Server, Oracle, MySQL

6. Finally, set the Section Footer as shown in the screenshot, and then click the Add Section button:



7. Now let's switch homes. In Edit Sections, click on the home symbol for the newly created home, and then delete the OldHome section. Here's your new home:



We'll take care of improving the looks later; but for now the functionality is there. After adding new sections, their links will be listed in the home page, instead of the boring This section has no child sections message.

About Me

Since this is a personal website, we do need an About Me page, don't we? Let's add one. Because we want this to be a simple page without any special functionality, we'll create it as an HTML page section.

1. Go to Community Administration Menu, and add a new section with the following properties:

Name	Value
Section Name	About
Section Title	About Me
Section Menu Title	ABOUT ME
Section Description	Let me do my best to answer the "Who am I?" question.
Section Type	HTML Page Section

2. Switch to the Appearance tab to set the Section Page Header, in HTML mode:

```
<SPAN CLASS="About_Normal">
  <B>Let me do </B> my best to answer to the <EM>"Who am I?"</EM> question.
</SPAN>
```

3. Click on Add Section and navigate to the About Me section in the main site. There—assuming you're still logged in as Admin—you'll see an Edit HTML link:



4. Click the Edit HTML link, and then check the View HTML Source checkbox to start editing in HTML mode:

```
<P ALIGN="CENTER">
  <IMG SRC="cristian2.jpg"/>
</P>
<SPAN CLASS="About_Normal">
  <B>I admit </B>this is a tough section to write. This is the part of the
  site where I should answer the <EM>"Who am I?"</EM> question.
<BR><BR>
  <B>Although </B>it is said that even the most complex questions have simple
  answers, I can't seem to find a simple answer to that particular question.
  So, let me just tell you the usual stuff about where I live, what I like,
  what I do for living and so on.
</SPAN>
```

HTML pages (including this one) are stored in the `Community_HtmlPages` table in the CSK database.

5. Click Apply changes. You're redirected to the About section.

Before being able to see the picture (`cristian2.jpg`), you need to add it to the database as a community image. Switch to the Community Administration Menu, go to the Edit Images page, and add there `cristian2.jpg`, which is available in the code download.

By default all requests for image files are handled by an image handler class named `ImageHandler`, which automatically searches for the image in the database (the `ImageHandler` class is registered in `web.config`). However, it is also possible to reference an image that resides in a physical folder on the disk, if that folder (or one of its parents) has an additional `Web.Config` file that unhooks the `ImageHandler` class (we'll see more about this in the following chapter). An example of such a folder is `CommunityStarterKit/Communities/Common/Images`.

When loading the About Me page in Internet Explorer, you'll now see:



Advertising the Books

We need to add some books to our site. To keep things simple, we'll add a single book. This doesn't change the fact that we do need a Books section. Here are the steps you need to follow:

1. Make another trip to the Community Administration Menu and add a new section with the following:

Name	Value
Section Name	Books
Section Title	My Books
Section Menu Title	MY BOOKS
Section Description	Interesting details, downloads and other information related to my books.
Section Type	Book Section

2. In the Appearance tab, set the HTML of Section Page Header like this:
Knowledge is power, and very often knowledge comes through books. It is a great pleasure for me to share what I know with my fellow programmers,

especially when I'm not doing it for free:) My books are in the area of Microsoft .NET programming, and databases. Please click on any of the links below to find more details about a particular book.

3. Click Add Section. The Books section will appear as in the following image:



4. Click the Add Book link to add a new book:

Name	Value
Book Title	Beginning ASP.NET E-Commerce, 2nd Edition
Author	Cristian Darie, Karli Watson
ISBN	1590592549
Purchase Link	http://www.cristiandarie.ro/eCommerceBook2.html
Price	39.99
Book Brief Description	The excellent feedback we received for the first edition inspired us to produce an even better book for those of you starting to build e-commerce solutions with ASP.NET, VB.NET and SQL Server.
Book Full Description	The second edition features improved code, more theory, more code, and more tips & tricks about how to build effective e-commerce sites! The book was also updated for latest versions of .NET and Visual Studio .NET.

5. The results are as follows:



Unlike with the image in the HTML page (which we added as a community image to the database), here the image file was automatically added to the database as a section image.

Hey, we now have a Books section! Authenticated users can even add comments by default on the books, but you are always free to change the security options. Don't forget that you can also enable ratings!

Forum

The final section we're adding is the forum, which is probably the feature that makes this website a true community website. Here's the path to make it live:

1. Make another trip to the Community Administration Menu and add a new section:

Name	Value
Section Name	Forum

Section Title	Forum
Section Menu Title	FORUM
Section Description	Feel free to discuss about any decent topics here
Section Type	Discuss Section

-
2. In the Security tab, add Community-Authenticated to Add Roles. This will allow any authenticated user, not only the administrators, to post to the forum.
3. Press the Add Section button. Congratulations, you have a forum on the website!

Changing the Layout

If you're as excited as I am about how easy it has been to build the site so far, you're probably curious about what's next. Well, it's time to change the colors.

Updating the Parent Section and the Styles

ParentSection is the module that generates the Home page. Apart from changing the styles used to display text, we want to get rid of the HOME title that appears at the top of the page. Apart from modifying Home, we'll add or change a number of styles that update the look of various parts of the website. Let's do this now.

1. In order to remove the title from ParentSection, we need to change the skin file ParentSection_ParentSections.ascx. Because we don't want to mess with the default theme, we will copy this file from the default theme to our Blue theme.
2. In Visual Studio, copy ParentSection_ParentSection.ascx from Communities/Common/Themes/Default/Skins/ContentSkins to Communities/Common/Themes/Blue/Skins/ContentSkins, and choose no when asked about creating a new class. This way, the Blue theme will use its own copy of the file, instead of the version from the default theme.
3. Open the ParentSection_ParentSection.ascx from the Blue theme in Design view. It has four controls: SectionTitle, PageHeader, ContentList, and PageFooter. Delete the SectionTitle (the one that generates the Home title).
4. Open the Default.css file of the Blue theme and modify these styles:

```
.BODY
{
    scrollbar-face-color: #336182;
    scrollbar-highlight-color: #FFFFFF;
    scrollbar-shadow-color: #FFFFFF;
    scrollbar-3dlight-color: #336182;
    scrollbar-arrow-color: #FFFFFF;
    scrollbar-track-color: #d3d3d3;
    scrollbar-darkshadow-color: #336182;
}

.FooterText
{
    font-family: Verdana;
    font-size: 10px;
```

```

    color: #ffffff;
}

.ParentSection_SectionDescription
{
    font-size: 14px;
    font-family: 'Microsoft Sans Serif';
}

.ParentSection_ListTitleLink
{
    font-weight: bold;
    font-size: 14px;
    font-family: 'Microsoft Sans Serif';
}

.ParentSection_ListBriefDescription
{
    font-size: 14px;
    font-family: 'Microsoft Sans Serif';
}

```

5. Also in Default.css, add the following style:

```

.Home_Header
{
    font-size: 14px;
    font-family: 'Microsoft Sans Serif';
}

```

6. After these changes, the look of the Home page changes a little bit:



7. Continue by modifying the following two styles used in the Books section:

```

.BookSection_SectionTitle

```

```
{
  font-weight: bold;
  font-size: 17px;
  line-height: 30pt;
  font-family: 'Microsoft Sans Serif';
  font-variant: small-caps;
}
```

```
.BookSection_SectionDescription
{
  font-size: 14px;
  font-family: 'Microsoft Sans Serif';
}
```

8. Add the following style, which defines the text that appears in the About Me page:

```
/* Text in the About page */
.About_Normal
{
  font-size: 14px;
  font-family: 'Microsoft Sans Serif';
}
```

9. Add the following style, which is needed in the layout we build in the next section:

```
.text_mik3
{
  font-size: 5px;
  color: #ffffff;
  font-family: Verdana;
}
```

You need to add the images in the Images folder of the Blue theme folder. Right-click on the Images folder, select Add Existing Item, and browse to the folder where you downloaded the image files (you'll need to specify *.* in the file types box to see the image files). Select all existing files and press Open. Your Blue folder should now contain the following items as shown in the following screenshot:



Let's change the layout. Yes, the code listing is a bit long, but it's all simple HTML with a few CSK-specific bits listed here:

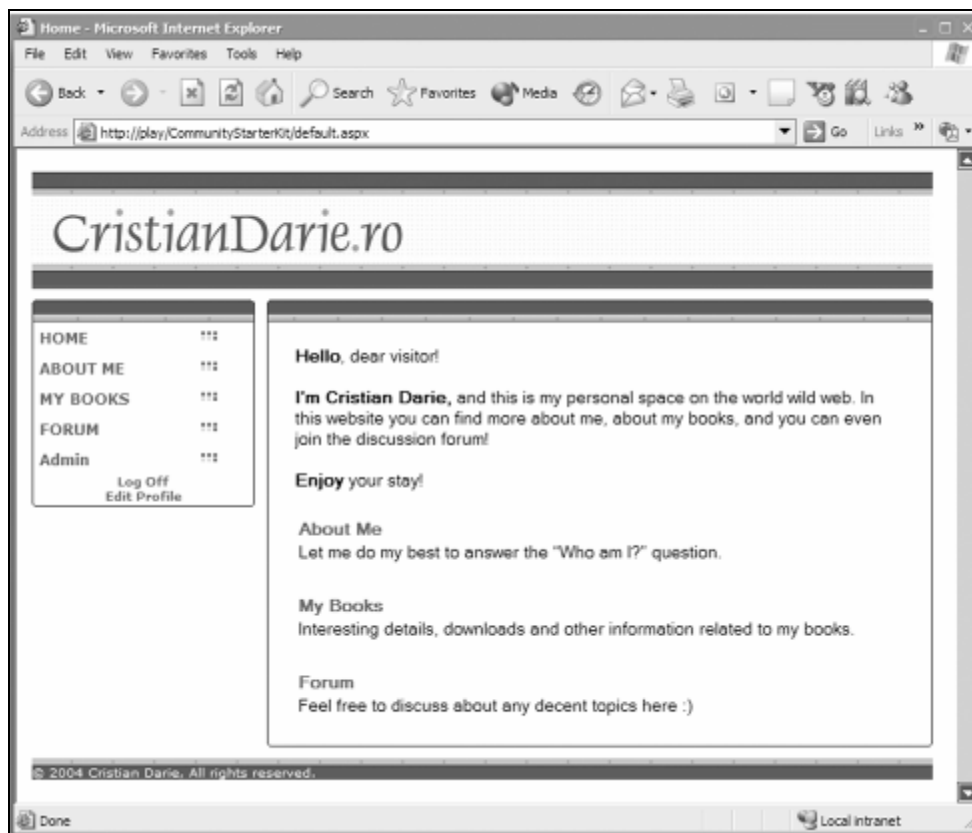
- The <community:SectionMenu> element, which generates the section menu
- The <community:UserLogin> element, which generates the login/logout controls
- The <asp:Placeholder id="Content"> element, which marks the site's content area
- The <asp:Placeholder id="Footer"> element, which marks the page footer

In the existing page skins, it's easy to determine which server controls are used in code by looking after their Runat="Server" property.

The complete code for the new default.ascx skin is provided in the code bundle.

```
<community:SectionMenu CssClass="SectionMenu" Runat="server"
    id="SectionMenu1" />
<community:UserLogin CssClass="UserLogin" Runat="server" id="UserLogin1" />
<asp:Placeholder id="Content" Runat="Server" />
<asp:Placeholder id="Footer" Runat="Server" />
```

You've done it! After changing the layout as shown, the website should look like this:



Summary

This little exercise demonstrated how efficient it is to build or rebuild websites using the Community Starter Kit. With minimal amount of work, we've got a completely functional website that allows visitors to register, participate on the forum, add comments on existing content, and so on. Yes, the possibilities are exciting!

For those of you that want to get even more from the CSK, we shall now focus on how the CSK is designed and how to code additional functionality into it.

6

The Community Starter Kit Core Architecture

Understanding the core architecture of the **Community Starter Kit (CSK)** is essential if you need to extend or change it. In this chapter, we'll learn how the CSK builds the client pages, and what are the major classes that make things work. In later chapters, we'll take advantage of the knowledge gained in this chapter to build our own CSK modules.

In this chapter we will:

- Learn more about the CSK folder structure and its configuration files
- Understand how the CSK engine builds the client pages
- Learn about a number of important middle-tier classes
- Look at the database structure

The CSK Folder Structure and Configuration Files

Learning about what each folder in the CSK project does will make things easier to digest in the latter part of the chapter, where we get down to details. Fortunately, the CSK folders are organized in a logical manner in the CSK project.

The following screenshot shows the files and folders in the CSK root folder. You've seen some of these in the previous chapters.



Configuring the CSK Project

We start by learning about the `web.config` configuration file and the ASP.NET application file (`Global.asax`). Then we'll learn about the root folders in the CSK project.

In this chapter we'll make frequent references to the data tables and stored procedures in the CSK database. If you don't have access to SQL Server's Enterprise Manager (in other words, if you're working with MSDE), you can use the Server Explorer window in Visual Studio .NET. It permits you to view and edit the tables and stored procedures, but you shouldn't really modify anything directly in the database unless you know for sure what you're doing!

Web.Config

Much of the CSK functionality is related to `web.config`. This file begins with four groups of configuration elements:

```
<configuration>
  <configSections>
    <sectionGroup name="communityStarterKit">
      <section name="Isp"
        type="ASPNET.StarterKit.Communities.CommunitiesSectionHandler,
        ASPNET.StarterKit.Communities" />
      <section name="pagePaths"
        type="ASPNET.StarterKit.Communities.CommunitiesSectionHandler,
        ASPNET.StarterKit.Communities" />
      <section name="database"
        type="ASPNET.StarterKit.Communities.CommunitiesSectionHandler,
        ASPNET.StarterKit.Communities" />
      <section name="services"
        type="ASPNET.StarterKit.Communities.CommunitiesSectionHandler,
        ASPNET.StarterKit.Communities" />
    </sectionGroup>
  </configSections>
```

A configuration group is defined for easy access to your custom configuration sections, and ensures there are no conflicts with configuration sections implemented by other programmers.

We have a group named `communityStarterKit` with four sections defined: `Isp`, `pagePaths`, `database`, and `services`. When defining a new section, the `type` attribute of the section element contains the name of the configuration section handler class and its assembly name. In our case, the handler class is `CommunitiesSectionHandler`, which simply inherits from `System.Configuration.NameValueFileSectionHandler` without adding any functionality of its own.

After defining the group sections, you can access their values by calling `ConfigurationSettings.GetConfig("<group name>/<section name>")`, which returns a `NameValueCollection` instance containing the values we need.

ISP Administrator Information

`Web.Config` continues with the `communityStarterKit` element defined earlier as a section group, and within that is the `Isp` section. The `Isp` section contains the ISP username and password data, used to log in to the `ISPAdmin` page:

```
<communityStarterKit>
  <Isp>
    <add key="IspUsername" value="ISPAdmin" />
    <add key="IspPassword" value="ISPPassword" />
  </Isp>
```

Default Page Information

The `pagePaths` element contains the names of the files that handle web page and web service requests on a community website. CSK uses a single file to handle web page requests, and that file is named `communityDefault.aspx` by default. The similar file that responds to all web service requests is `communityService.asmx`. The names of these files can be configured here:

```
<pagePaths>
  <add key="basePage" value="/communityDefault.aspx" />
  <add key="baseService" value="/communityService.asmx" />
</pagePaths>
```

Database Connection Information

The `database` element contains the connection string for the CSK database:

```
<database>
  <add key="connectionString" value="Server=(local);
    Trusted_Connection=true;database=CommunityStarterKit" />
</database>
```

Enabling Web Services Timer

The `services` element sets `enableServiceTimer` to `true`, which enables the CSK to look for remote content at intervals of 15 minutes. This value is read and used in `Global.asax`.

```
<services>
  <add key="enableServiceTimer" value="true" />
</services>
</communityStarterKit>
```

Enabling Debug Mode

After the `communityStarterKit` element, we meet `system.web`, and its first node is the `compilation` element. By default, the debug mode is disabled, and it's very likely that you'll need to use the debug features while dissecting the code. To enable this mode, you need to modify `Web.Config` as follows:

```
<system.web>
  <compilation debug="true" />
  <customErrors mode="Off"/>
```

Handling Client Requests

The `httpModules` element registers the `CommunitiesModule` class, which will process all client requests, including the ones for web forms, web services, image files, and so on. Requests for ASPX and ASMX resources are redirected to the files configured in the `pagePaths` element for further processing.

```
<httpModules>
  <add name="CommunitiesModule"
        type="ASPNET.StarterKit.Communities.CommunitiesModule,
        ASPNET.StarterKit.Communities" />
</httpModules>
```

Configuring Authentication Information

The `authentication` element specifies the method to authenticate users. The authentication mode chosen is `Forms` (the other values are `Windows`, `Passport`, and `None`), and the (virtual) page to handle login requests is `Users_Login.aspx`:

```
<authentication mode="Forms">
  <forms loginUrl="Users_Login.aspx"/>
</authentication>
```

Setting the Maximum Request Length

The maximum request length limits the size of the files you can upload to the database, so you might want to set this to a higher value:

```
<!-- We need to make sure this is set high enough so that the user doesn't get
a timeout error while uploading. -->
<!-- The largest file size that ASP.NET will allow for uploads. -->
<httpRuntime
  maxRequestLength="10000" />
```

Handling Image File Requests

The next element of interest is `httpHandlers`, which registers the `ImageHandler` class to process all the requests for files with the extension `JPG`, `JPEG`, or `GIF`. When such a file is requested, the `ImageHandler` class generates the image file from the database (community and section images are stored in the database).

```
<httpHandlers>
  <add verb="*" path="*.jpg"
        type="ASPNET.StarterKit.Communities.ImageHandler,
        ASPNET.StarterKit.Communities" />
  <add verb="*" path="*.jpeg"
        type="ASPNET.StarterKit.Communities.ImageHandler,
        ASPNET.StarterKit.Communities" />
  <add verb="*" path="*.gif"
        type="ASPNET.StarterKit.Communities.ImageHandler,
        ASPNET.StarterKit.Communities" />
</httpHandlers>
```

Configuring Web Services

Finally, under the `<webServices>` element, the `HttpGet` protocol is enabled to interpret client requests:

```
<webServices>
  <protocols>
    <add name="HttpGet"/>
  </protocols>
</webServices>

</system.web>
</configuration>
```

The other three possible values are `HttpSoap`, `HttpPost`, and `Documentation` (more of these can be enabled by placing additional `add` elements).

Global.asax

The code-behind file of `Global.asax` implements the timers used for service subscriptions. It calls the `CreateTimers()` method in the `Application_Start` event handler, which executes at application startup.

In `CreateTimers()` a new `Timer` object is created. Through the `WebServiceTimer_Elapsed` event handler, this object ultimately calls `ServiceUtility.GetRemoteContent()`, which updates the database with subscribed content from remote sources.

```
private void CreateTimers() {
    if (CommunityGlobals.EnableServiceTimer) {
        System.Timers.Timer WebServiceTimer = new
        System.Timers.Timer();
        WebServiceTimer.Enabled = true;
        WebServiceTimer.Interval = (60000 * 15);
        WebServiceTimer.Elapsed += new
        System.Timers.ElapsedEventHandler(WebServiceTimer_Elapsed);
    }
}
```

The default interval is 15 minutes, but the update process only occurs if the value of `enableServiceTimer` in `Web.Config` is true (`CommunityGlobals.EnableServiceTimer` returns a boolean value representing the value of `enableServiceTimer`).

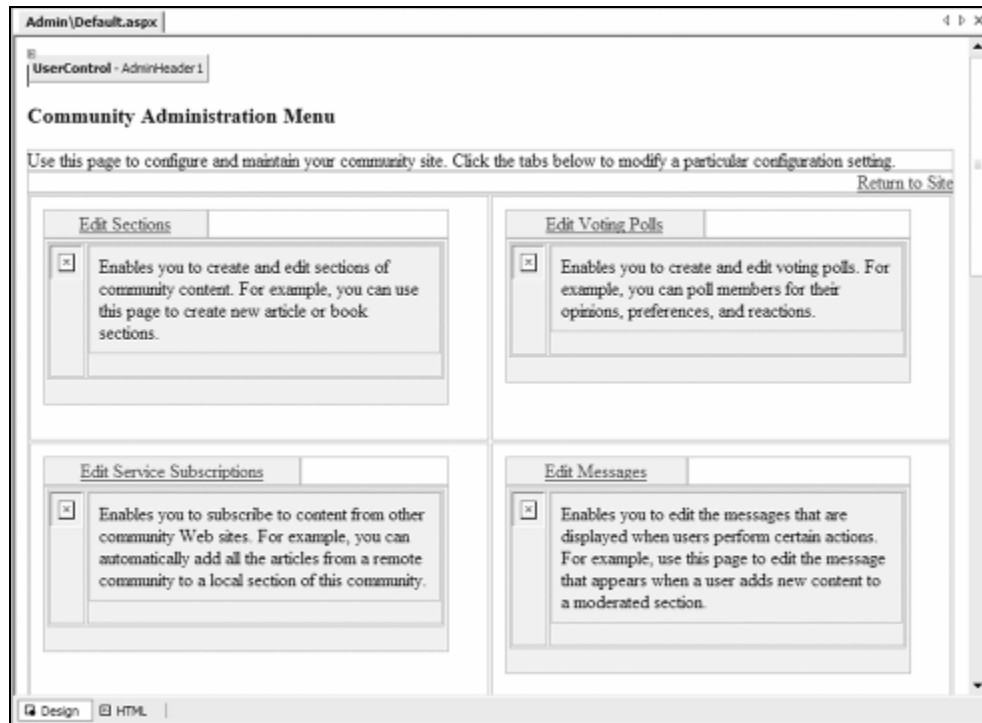
What Do the Main CSK Folders Contain?

The five main folders located in the root of the CSK project are: `Admin`, `Advertisements`, `Communities`, `ISPAdmin`, and `Engine`. Because much of the CSK design is based on the folder structure, here is an overview of each of these folders.

The Admin Folder

The `Admin` folder contains the pages that show up in the Community Administration Menu (we talked about this in Chapter 3).

This folder contains two files: `default.aspx` and `Web.Config`. The `default.aspx` file represents the main page of the Community Administration Menu. The following screenshot shows `default.aspx` in the Design view in Visual Studio .NET:



The Admin folder contains a number of subfolders (each of them having a `default.aspx` page, and optionally some web user controls), which implement the sections that exist in the Community Administration Menu main page. These subfolders are: Activity Log, Delete Content, View Documentation, Edit Images, Edit Messages, Edit Sections, Edit Service Subscriptions, Edit Topics, Edit Users, Edit Voting, Edit Web Box Services, and Send Newsletter. Apart from these, there are four additional subfolders in the Admin folder:

- controls: Contains a single user control named `AdminHeader.ascx`, representing the header that appears in all Community Administration Menu pages
- help: Contains the help file that appears when clicking on the Click here for a preview of all themes link, in Community Administration Menu | Edit Sections | Edit Section | Appearance
- Images: Contains images for the Community Administration Menu
- styles: Contains a CSS file named `Admin.css`, which provides the styles used in the Community Administration Menu

The `Web.Config` file in Admin defines a couple of configuration options.

The file is listed as follows:

```
<configuration>
  <system.web>
    <authorization>
      <allow roles="Community-Administrators"/>
      <deny users="*" />
    </authorization>

    <httpHandlers>
      <remove verb="*" path="*.jpg" />
      <remove verb="*" path="*.gif" />
    </httpHandlers>
  </system.web>
</configuration>
```

The first part of `web.Config` specifies the security roles allowed to access the community administration pages. By default only Community Administrators are allowed to access the Community Administration Menu.

The `httpHandlers` element revokes the effect of the `httpHandlers` element of the main `web.Config` file, which assigns the `ImageHandler` class to handle requests to `*.jpg` and `*.gif` files (note that the main `web.Config` file also mentions the `*.jpeg` extension). This is necessary because the image files for the Community Administration Menu are stored directly under the `Admin/Images` folder and don't need to be retrieved from the database through the `ImageHandler` class as with the other images in the community website.

The `CommunitiesModule` class, defined in `web.Config`, is used to process requests to the Admin pages as well.

You'll need to visit the `Admin` folder whenever you need to add or remove features from the Community Administration Menu.

The Advertisements Folder

The `Advertisements` folder contains information for the ads in the community website. We learned about them in Chapter 3.

The Communities Folder

The `Communities` folder contains themes, images, and web box controls (usually simply referred to as Web Boxes) required when building pages for the client.

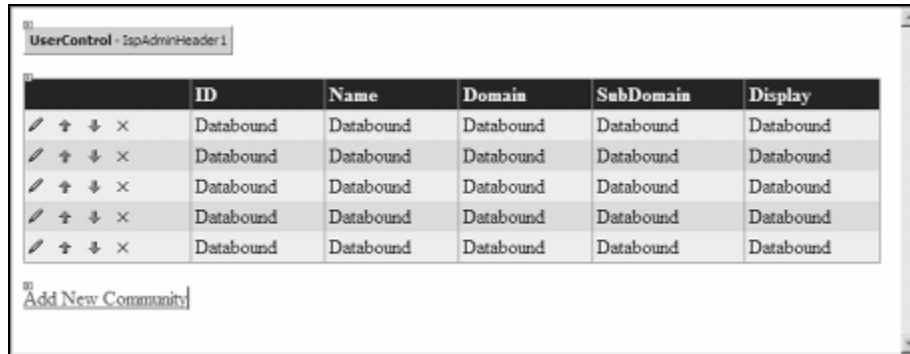
The `Communities` folder contains a subfolder named `Common` that contains data (themes, images, Web Boxes) visible from all communities. Optionally, you can find other subfolders containing community-specific themes, images, or Web Boxes.

We saw the `Communities` folder in Chapter 4 when we discussed themes and skins.

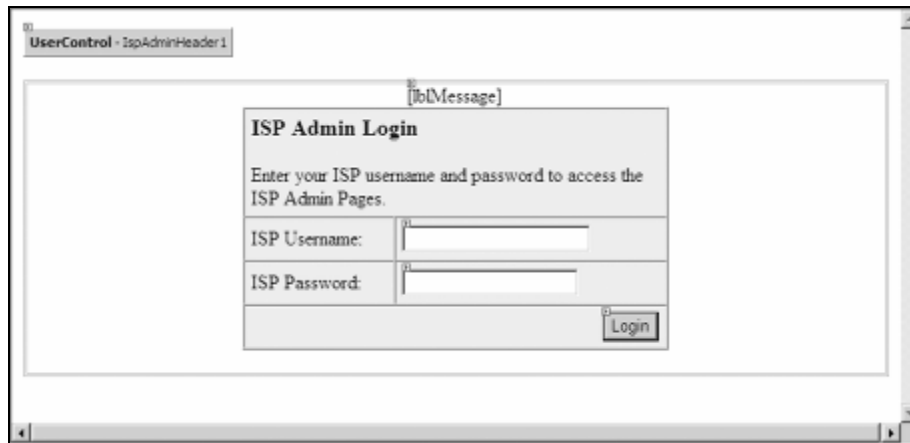
The ISPAdmin Folder

ISPAdmin contains the ISP Admin pages. It contains three files—`default.aspx`, `ISPLLogin.aspx`, and `web.Config`—and a number of subfolders.

`default.aspx` is the page that lists the available communities, and offers the means to delete, modify, or add new communities. The following screenshot shows `default.aspx` with its main Datagrid control in the Design view window in Visual Studio .NET:



`ISPLLogin.aspx` is the page that requests login information for ISP administrators:



The `web.Config` is listed as follows:

```
<configuration>
<system.web>
  <httpModules>
    <remove name="CommunitiesModule"/>
  </httpModules>
```

```

<httpHandlers>
  <remove verb="*" path="*.jpg" />
  <remove verb="*" path="*.gif" />
</httpHandlers>
</system.web>

</configuration>

```

Like the `Web.Config` file in `Admin`, it contains an `httpHandlers` element that revokes the effect of the `httpHandlers` element defined in the main `Web.Config` file. It also has an `httpModules` element that unhooks the `ISPAdmin` folder from the `CommunitiesModule` class.

`ISPAdmin` also contains the following folders:

- `AddCommunity` and `EditCommunity` contain web forms for the Add Community and Edit Community pages. Open the default.aspx page inside these folders to see how this works. The actual functionality is performed by methods in the `CommunityUtility` class.
- `Controls` contains the `ISPAdminHeader.ascx` control, which represents the header that appears in every page in the ISP Admin.
- `Images` contains the image files for the ISP Admin pages.
- `SampleData` contains two web forms: `ExportSampleData.aspx` and `ImportSampleData.aspx`. Because they aren't linked to other CSK web pages, you can only access these by directly typing their address, such as `http://localhost/CommunityStarterKit/ISPAdmin/SampleData/ExportSampleData.aspx`. Here's how `ExportSampleData.aspx` looks in Design view (`ImportSampleData.aspx` looks similar):



The `Export` button in `ExportSampleData` calls `CommunityUtility.ExportSampleData`, saving a number of XML files (such as `Community_SampleArticles.xml` and `Community_SampleBooks.xml`) containing the sample data of the Lunar .NET community to the `SampleFiles/SampleFiles` folder. The sample data is retrieved from the database.

The import page has reverse functionality. It updates the data in the database sample tables according to the information in the XML files.

The Engine Folder

This folder contains most of the functionality that makes the CSK work. It contains the `AssemblyInfo.cs` file and the following three subfolders:

- `Admin` contains the functionality that supports the Community Administration Menu and ISP Admin pages.
- `Modules` contains functionality that supports each CSK module (every module representing a content type such as articles, books, and so on).
- `Framework` contains the base classes that support all the existing functionality in the CSK, including the module pages.

Building Pages for the Client

An interesting aspect of the CSK is the way it responds to client requests, based on "fictional" virtual locations, such as:

`http://localhost/CommunityStarterKit/Favorite+Links/default.aspx`.

The web location for most community pages refer to files or resources that don't physically exist at the mentioned locations, or don't even exist at all in the file system. All requests are automatically forwarded to a single web form named `communityDefault.aspx`, which builds a response for the client.

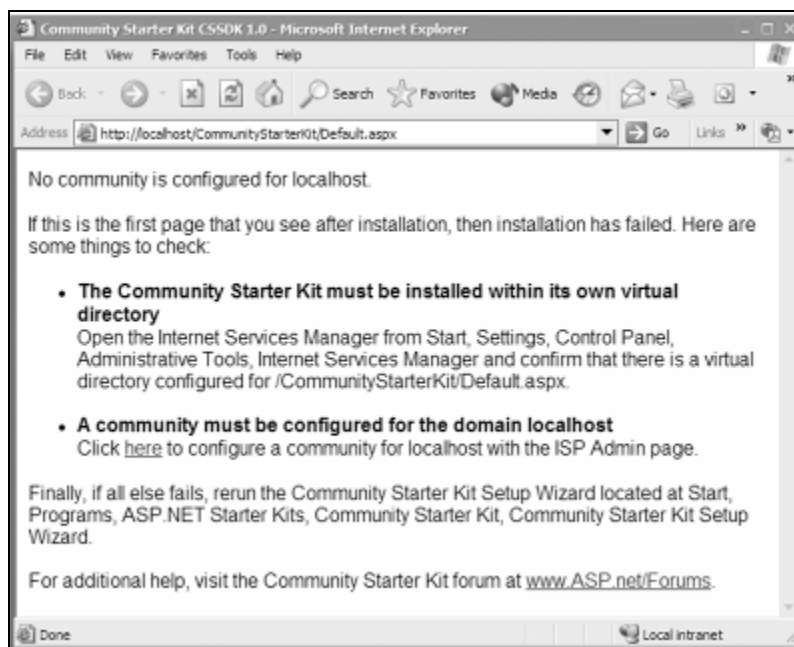
On every request for a community web page, two important actions occur:

10. The request is processed by a class named `CommunitiesModule` (configured in `web.Config`). This class extracts the information about the page or resource being requested and saves it into the `Context` in the form of a `SectionInfo` object. Note that requests to image files are finally processed by another class, named `ImageHandler`, which is also configured in `web.Config`.
11. Then, `CommunitiesModule` redirects the request to `communityDefault.aspx` (or `communityService.asmx` for web service requests), no matter which resource was initially asked for. This file will know what to show the visitor based on the info it gets from the `Context`.

The `HttpContext.Cache` object is frequently used to cache information retrieved from the SQL Server database. This improves performance significantly, since the cache performs much faster than the database.

The Default Page

If something goes wrong in the code that redirects to `communityDefault.aspx`, or if there are no communities defined in the CSK installation, the response will be built by the "real" default page, `default.aspx`. This page has no code and only displays an error message saying that no communities are configured:



If you want to see this page, you should either insert a return statement at the beginning of the `Application_BeginRequest` method in the `CommunitiesModule` class (which we'll see a bit later), or use the ISP Admin page and remove all the communities in your CSK installation. Don't try the latter method if your database contains important data, as this will cause all the existing data to be deleted!

Analyzing the Community Default Page

`communityDefault.aspx` doesn't have any HTML code itself, but its code-behind file uses the skin files of the current theme to generate content for the page being visited. You can refer to Chapter 4 to review themes and skins.

The method that generates the client output in `communitydefault.ascx.cs` is `Page_Init`. It looks quite frightening at the first look, but its functionality is simple.

The `Page_Init` method performs its duties by using a number of CSK base classes. We'll discuss these classes later in this chapter; for now, we'll focus on understanding the general process through which a community web page is built for the client.

First let's see the HTML code that the `Page_Init` method generates—this will help understanding how it works.

Here's a sample output from the Favorite Links page:

```
<html>
  <head>
    <title>Favorite Links</title>
    <link href=
      "/CommunityStarterKit/Communities/Common/Themes/Default/Styles/Lunar.css"
      type="text/css" rel="stylesheet"/>
    <META NAME="KEYWORDS" CONTENT=""/>
    <META NAME="DESCRIPTION" CONTENT=""/>
    <META NAME="PAGETYPE"
      CONTENT="ASPNET.StarterKit.Communities.Links.LinksSection"/>
    <META ASPNETSTARTERKIT="CommunitySK" Version="CSVS 1.0"/>
  </head>
  <body marginheight="0" marginwidth="0" leftmargin="0" rightmargin="0"
  class="pageBody">
    <form name="PageForm" method="post" action="default.aspx" id="PageForm">
      ... contents here ...
      ... contents here ...
      ... contents here ...
    </form>
  </body>
</html>
```

Page_Init first constructs the head tag and generates its contents. Then it generates the form element and fills it with the page skin of the community, which is populated with a content skin in its content Placeholder. Let's see how all these actions take place, by looking at Page_Init line by line. At the beginning of the method, there are some initialization procedures:

```
void Page_Init(Object s, EventArgs e){
    Control objPageSkin = null;
    Placeholder objPagePart = null;

    // Rewrite Path Back to Original
    Context.RewritePath( Path.GetFileName( Request.RawUrl ) );

    // Get the PageInfo and SectionInfo objects
    PageInfo objPageInfo = (PageInfo)Context.Items[ "PageInfo" ];
    SectionInfo objSectionInfo = (SectionInfo)Context.Items[ "SectionInfo" ];
```

The objPageSkin object will represent the page skin of the community. objPagePart will be used to reference and populate two Placeholder objects: the main content zone of the page skin, and the page footer. Recall from Chapter 4 that the page skin is a web user control named Default.ascx located in the Communities/Common/Themes/theme_name/Skins/PageSkins folder.

objPageInfo is an object of type PageInfo (found in Engine/Framework/BaseClasses/PageInfo.cs), which stores information about the page being requested. objSectionInfo is a SectionInfo object (found in Engine/Framework/Sections/Components/SectionInfo.cs), which represents a community section. For now, you need to know that PageInfo stores information about the page being created as a whole (such as its title) and SectionInfo represents the details of the website section being visited.

Next, a StringBuilder object, named objBuilder, is created to store the HTML output that will be sent to the client. The StringBuilder class is used because it is much faster than string when concatenating strings. First, the head portion of the page is added to objBuilder:

```
// Create a StringBuilder to build up the Page Head
StringBuilder objBuilder = new StringBuilder();
objBuilder.Append( "<html>\n<head>" );
```

After creating the object and appending the html and head tags, the content of the head element is appended to the StringBuilder object. The Title of the page is extracted from the PageInfo object:

```
// Add the Page Title
objBuilder.AppendFormat( "\n<title>{0}</title>", objPageInfo.Title );
```

The path to the CSS file is extracted from the SectionInfo object (you can set the theme from Community Administration Menu | Edit Sections | Edit Section | Appearance):

```
// Add the Page Style
objBuilder.AppendFormat( "\n<link href=\"{0}\" type=\"text/css\"
    rel=\"stylesheet\"/>", CommunityGlobals.ResolveBase(
    "Communities" + "/" + objSectionInfo.Style ) );
```

The META tags are also populated with information from the PageInfo object. The fact that CSK automatically generates these tags comes handy in a web world striving for search engine placement.

```
// Add the Meta Tags
objBuilder.AppendFormat( "\n<META NAME=\"KEYWORDS\" CONTENT=\"{0}\"/>",
    Server.HtmlEncode(objPageInfo.MetaKeys) );
objBuilder.AppendFormat( "\n<META NAME=\"DESCRIPTION\" CONTENT=\"{0}\"/>",
    Server.HtmlEncode(objPageInfo.MetaDesc) );
objBuilder.AppendFormat( "\n<META NAME=\"PAGETYPE\" CONTENT=\"{0}\"/>",
    objPageInfo.Content );
objBuilder.Append( "\n<META ASPNETSTARTERKIT=\"CommunitySK\"
    Version=\"CSVS 1.0\"/>" );
```

Finally, the contents of objBuilder are added to the Page, constructing the page head. Here, the role of objBuilder is over.

```
// Add the Page Head to the Page
objBuilder.Append( "\n</head>\n<body marginheight=\"0\" marginwidth=\"0\"
    leftmargin=\"0\" rightmargin=\"0\" class=\"pageBody\">\n" );
Controls.Add( new LiteralControl( objBuilder.ToString() ) );
```

Next, an HtmlForm object is constructed. This will also be added to the Page at the end of the method, but not before being populated with the page skin:

```
// Create a Form Control
HtmlForm objForm = new HtmlForm();
objForm.ID = "PageForm";
```

The Page Skin object is initialized using the LoadControl function; the name of the currently selected skin is extracted from the objSectionInfo object:

```
// Load the Page Skin
objPageSkin = LoadControl( CommunityGlobals.AppPath + "/Communities/" +
    objSectionInfo.Skin + "/Skins/PageSkins/Default.ascx" );
```

After loading the page skin, the next important step is to fill its contents Placeholder. The Placeholder object in the page skins is named content:

```
// Add the Page Content Page Part
objPagePart = (Placeholder)objPageSkin.FindControl( "content" );
```

The Placeholder is populated with a control whose fully qualified name is retrieved through the PageInfo object:

```
if (objPagePart != null) {
    Control objPageContent = (Control)Activator.CreateInstance(
        Type.GetType( objPageInfo.Content, true ) );
    objPagePart.Controls.Add( objPageContent );
}
```

objPageInfo.Content is a string property of PageInfo that returns the fully qualified name of the custom control that fills the page content; examples of such controls are ASPNet.StarterKit.Communities.ParentSection.ParentSection (this is used for parent sections, including Home) and ASPNet.StarterKit.Communities.Articles.ArticleSection. Most of these content classes are located in /Engine/Modules/module_name/Content folders and generate their output by importing the layout from the content skins – we'll see more about these classes later. The fully qualified names of these classes are stored in the Community_PageTypes table in the CommunityStarterKit database.

Next, the search textbox is taken care of. A method named RedirectToSearchPage is designated to react to the TextChanged event (which fires when the *Enter* key is pressed). RedirectToSearchPage sends the visitor to a virtual page named Search_Search.aspx, which handles the search.

```
// Find Search
TextBox txtSearch = (TextBox)objPageSkin.FindControl("txtSearch");
if (txtSearch != null)
    txtSearch.TextChanged += new EventHandler(RedirectToSearchPage);
LinkButton lnkSearch = (LinkButton)objPageSkin.FindControl("lnkSearch");
if (lnkSearch != null)
    lnkSearch.CausesValidation = false;
```

Then, the page footer (taken from the objSectionInfo object) is added to the Footer Placeholder from the page skin:

```
// Add the Page Footer
objPagePart = (Placeholder)objPageSkin.FindControl( "Footer" );
if (objPagePart != null)
    objPagePart.Controls.Add( new LiteralControl( objSectionInfo.Footer )
);
```

At the end, objPageSkin is added to the objForm control, which is added to the page. The </body> and </html> tags are also added.

```
// Add the Page Skin to the Form
objForm.Controls.Add( objPageSkin );

// Add the Form Control
Controls.Add( objForm );

// Close the Page
Controls.Add( new LiteralControl( "\n</body>\n</html>" ) );
}
```

The last piece of code in communityDefault.aspx.cs is the RedirectToSearchPage, which executes when the visitor uses the search feature.

The response is redirected to a (virtual) page named Search_Search.aspx, with a search parameter in the query string whose value is the search string.

```

public void RedirectToSearchPage(Object s, EventArgs e) {
    string searchText = ((TextBox)s).Text;
    string redir = CommunityGlobals.CalculatePath(
        String.Format("Search_Search.aspx?search={0}",
            Context.Server.UrlEncode(searchText)));
    Page.SmartNavigation = false;
    Context.Response.Redirect(redir);
}

```

Processing Client Requests

The CSK implements the virtual folder redirection using a custom HTTP module—a class named `CommunitiesModule` (located in `Engine/Framework/BaseClasses/CommunitiesModule.cs`).

`CommunitiesModule` is the class that intercepts client requests and deals with them. We met this class earlier in this chapter, and we saw that it forwards all web form requests to `communityDefault.aspx` and web service requests to `communityService.asmx`.

`CommunitiesModule` implements the `IHttpModule` interface, which defines the `Init` and `Dispose` methods. The `Init` method in `CommunitiesModule` registers the two event handlers, `Application_BeginRequest` and `Application_AuthenticateRequest`, to handle the `BeginRequest` and `AuthenticateRequest` events:

```

public class CommunitiesModule : IHttpModule {
    public void Init(HttpApplication application) {
        application.BeginRequest += (new
            EventHandler(this.Application_BeginRequest));
        application.AuthenticateRequest += (new
            EventHandler(this.Application_AuthenticateRequest));
    }
}

```

Apart from `Init`, `CommunitiesModule` contains the following three methods:

- `Application_BeginRequest`
- `GetPageInfo`
- `Application_AuthenticateRequest`

Processing the Request in `Application_BeginRequest`

`BeginRequest` is the event that fires with the HTTP request. Its event handling method, `Application_BeginRequest`, gets called when any resource is requested from the server—no matter if it is an ASPX page or a JPEG picture.

Let's go step-by-step through `Application_BeginRequest` to see how it works. The call starts by saving a reference to the current `HttpContext` to an object named `Context`:

```

private void Application_BeginRequest(Object source, EventArgs e) {
    // Get abbreviated reference to current context
    HttpContext Context = HttpContext.Current;
}

```

The method sets the culture for the current thread based on the language of the visitor's browser.

This will influence the way date, time, and currency information will be displayed to the visitor.

```
// set culture based on acceptlanguage
if (Context.Request.UserLanguages != null) {
    try {
        Thread.CurrentThread.CurrentCulture =
            CultureInfo.CreateSpecificCulture(
                Context.Request.UserLanguages[0]);
    } catch {}
}
```

Next `requestPath` and `requestBasePath` are initialized:

```
// Retrieve path
string requestPath = CommunityGlobals.RemovePathInfo(
    Context.Request.Path.ToLower());
string requestBasePath = CommunityGlobals.GetSectionPath(
    requestPath);
```

`requestPath` will contain the path—such as `/communitystarterkit/mission/default.aspx` or `/CommunityStarterKit/Photo+Gallery/14.jpg`—to the requested resource. The `requestBasePath` is the base folder for the request, such as `/communitystarterkit/mission/`.

Next, `CommunityInfo` and `SectionInfo` objects (named `objCommunityInfo` and `objSectionInfo`) are created by calling two helper classes, and then stored in the `Context`. We'll discuss more details about these classes later; for now, you need to know that `CommunityInfo` contains information about the current community, such as its name, database quota, and the other details you establish when creating a new community in the ISP Admin page.

`SectionInfo` contains a long list of members describing the current section, such as its name, title, menu title, type, whether it has comments or topics enabled, and the roles that are allowed to perform various actions, and so on. Simply put, `SectionInfo` stores all the information you can set in the Community Administration Menu when creating or editing a community section.

No matter what resource is requested, it must have an associated `SectionInfo` object and an associated `CommunityInfo` object. `communityDefault.aspx` itself needs the `SectionInfo` object to generate the client output as we saw when discussing `communityDefault.aspx`.

If no community is configured in the CSK installation, `objCommunityInfo` will always be `null`; in this case, because the redirect to `communityDefault.aspx` never happens, the `default.aspx` page loads showing an error message.

The `CommunityInfo` and `SectionInfo` objects are generated by `CommunityUtility` and `SectionUtility` helper classes, which get the necessary data from the database:

```
// Figure out the community and add to Context
CommunityInfo objCommunityInfo =
    CommunityUtility.GetCommunityInfo();
if (objCommunityInfo == null)
    return;
Context.Items[ "CommunityInfo" ] = objCommunityInfo;

// Figure out the section and add to context
SectionInfo objSectionInfo =
    SectionUtility.GetSectionInfoFromPath(requestBasePath);
if (objSectionInfo == null)
    return;
Context.Items[ "SectionInfo" ] = objSectionInfo;
```

At the end of the `Application_BeginRequest` method, requests for ASPX pages are redirected to `communityDefault.aspx` and requests for ASMX web services are redirected to `communityService.asmx`. For other kinds of resources (such as image files), the method terminates normally without any redirection.

Remember that `web.Config` registers a class named `ImageHandler` that further processes requests for image files. This is necessary because many community images are in fact stored in the database instead of the Windows file system. The folders that contain image files in the file system unhook them from the `ImageHandler` class through local `web.Config` files.

The default value returned by `CommunityGlobals.UrlBaseService` is `/communityService.asmx`, and the default value returned by `CommunityGlobals.UrlBasePage` is `/communityDefault.aspx`. These file names are stored in `web.Config`, and are loaded with a call to `Context.RewritePath`:

```
// Is this a Web Service request?
if (requestPath.EndsWith(".asmx")) {
    Context.RewritePath(CommunityGlobals.UrlBaseService +
        Context.Request.PathInfo);
    return;
}

// Figure out the page and add to Context
if (requestPath.EndsWith(".aspx")) {
    PageInfo objPageInfo = GetPageInfo(objSectionInfo,
        requestPath);
    if (objPageInfo != null) {
        Context.Items[ "PageInfo" ] = objPageInfo;
        Context.RewritePath(CommunityGlobals.UrlBasePage);
    }
}
```

In case of ASPX resources, before redirecting to `communityDefault.aspx`, a `PageInfo` object named `objPageInfo` is also created and saved to the `Context`. The `PageInfo` object is created in a separate method named `GetPageInfo` in `CommunitiesModule`; we'll look at this next. `PageInfo` contains a number of members that describe a community page (such as the page title and description), and are used by `communityDefault.aspx` in the process of building a response for the client.

Getting Page Information in `GetPageInfo`

`GetPageInfo`, the method that returns a `PageInfo` object based on the current `SectionInfo` object and request path, is worth having a closer look at. This is because it generates a `PageInfo` object depending on the kind of ASPX page being requested.

The CSK recognizes three kinds of virtual ASPX pages:

- **default.aspx** pages are the main content pages of website sections, including pages like `http://localhost/CommunityStarterKit/Mission/default.aspx`, or the main `default.aspx` page.

- **content pages** are pages that represent a particular content item, such as an article or a book. These pages have numerical names, a typical example being something like `http://localhost/communitystarterkit/mission/49.aspx`.
- **named pages** are pages that don't have numerical names, and aren't `default.aspx` pages either. Typical examples of these pages are the ones that edit content items (`/Mission/Articles_EditArticle.aspx?id=49`), delete content items (`/Mission/ContentPages_DeleteContentPage.aspx?id=49`), or add content items (`/Mission/Articles_AddArticle.aspx`). Other named pages are the login (`users_login.aspx`), logout, register, and the search (`Search_Search.aspx`) pages among others.

An important difference between these page types is where their information is stored in the database. Analyzing the `GetPageInfo` method reveals this secret. Simply put, `GetPageInfo` performs these steps:

12. First, `GetPageInfo` checks if the requested page is a `default.aspx` page. Because `default.aspx` pages are the main section pages, `GetPageInfo` is able to create a `PageInfo` object based solely on the information it extracts from the `SectionInfo` object associated with the page being requested. The `requestFile` variable is initialized at the beginning of `GetPageInfo` with the name of the file that was requested.

```
// Check for section default page
if (requestFile.ToLower() == "/default.aspx")
    return new PageInfo
    (
        -1,
        objSectionInfo.Name,
        objSectionInfo.ParentSectionID,
        objSectionInfo.PageType,
        objSectionInfo.Title,
        objSectionInfo.Description,
        objSectionInfo.PageMetaDesc,
        objSectionInfo.PageMetaKeys,
        objSectionInfo.Content
    );
```

The ID of `default.aspx` pages is -1. For the content pages and named pages this number represents the ID that identifies the mentioned page in the database.

13. If the requested file is not a `default.aspx` page, `GetPageInfo` checks if it is a named page by calling `NamedPageUtility.GetPageInfoFromNamedPage`. If the method returns a valid `PageInfo` object, the page is indeed a named page.

```
// Check for named page
objPageInfo =
    NamedPageUtility.GetPageInfoFromNamedPage(requestFile);
if (objPageInfo != null)
    return objPageInfo;
```

`NamedPageUtility.GetPageInfoFromNamedPage` gets the necessary information by reading the `Community_NamedPages` table in the database, which stores information about

all the community named pages. You might want to open this table to see the other details it contains; we'll discuss more about the CSK database later in this chapter.

14. If the page wasn't a named page, `GetPageInfo` continues by assuming the page is a content page. It calls `ContentPageUtility.GetPageInfoFromPageContentID` to obtain the `PageInfo` object for the content page being requested:

```
// Check for content page
objPageInfo = ContentPageUtility.GetPageInfoFromPageContentID
(requestFile, objSectionInfo);
if (objPageInfo != null)
    return objPageInfo;
```

`ContentPageUtility.GetPageInfoFromPageContentID` gets the necessary information by reading the `Community_PageTypes` and `Community_ContentPages` tables in the database.

15. Finally, if none of the previous methods returned a `PageInfo` object, the method returns null.

```
// If everything fails, just return null
return null;
}
```

Perhaps the most important information `PageInfo` contains regarding the requested ASPX file is a string property named `Content`, which contains the fully qualified name of the class representing the page, such as `ASPNET.StarterKit.Communities.ParentSection.ParentSection` or `ASPNET.StarterKit.Communities.Articles.ArticleSection`.

Here again is the code snippet from `communityDefault.aspx.cs` that creates a control class based on the `PageInfo.Content` property:

```
if (objPagePart != null) {
    Control objPageContent = (Control)Activator.CreateInstance(
        Type.GetType( objPageInfo.Content, true ) );
    objPagePart.Controls.Add( objPageContent );
}
```

The class names stored in `PageInfo.Content` are located in various places in the CSK engine, and we'll learn later in this chapter how they work and how they generate their output based on the skin files.

Now let's continue with the last event handling method in `CommunitiesModule`, the one which authenticates client requests.

Authenticating the Request in `Application_AuthenticateRequest`

Each request is authenticated in the `Application_AuthenticateRequest` event handler (we're still in the `CommunitiesModule` class). This method verifies if the currently logged-in user is allowed to

access the requested resource, no matter if we're talking about a web page, a web service, or an image.

The function begins by calling `UserUtility.GetUserRoles`, which populates `Context.User` with a custom principal that contains the roles associated with the current user:

```
void Application_AuthenticateRequest(Object sender, EventArgs e) {
    HttpContext Context = HttpContext.Current;

    // Get the user roles
    UserUtility.GetUserRoles();
}
```

If the user initiating the request isn't authenticated, `UserUtility.GetUserRoles()` doesn't do anything. Internally, the function first gets the roles information from the database and stores it in an encrypted cookie on the client browser. On subsequent calls, the data is read directly from the cookie.

Next, a `UserInfo` object is created. The `UserInfo` object contains information about the permissions the user has in the section being visited, internally stored in private boolean members named `_mayView`, `_mayAdd`, `_mayEdit`, `_mayDelete`, `_mayComment`, `_mayModerate`, and `_mayRate`. The names are self-describing. Because `UserInfo` contains permissions related to a particular section, its single constructor requires a `SectionInfo` object as parameter:

```
// Get SectionInfo
SectionInfo objSectionInfo = (SectionInfo)Context.Items[ "SectionInfo" ];
if (objSectionInfo == null)
    return;

// Create the User Info Object
UserInfo objUserInfo = new UserInfo(objSectionInfo);
```

Once the `UserInfo` object is created, it's saved to the context, from where it can be retrieved within the application.

```
Context.Items[ "UserInfo" ] = objUserInfo;
```

The `UserInfo` class has a very useful method named `IsInRole`, which returns a boolean value indicating whether a user is in the specified role, and also recognizes the generic `Everyone` and `Authenticated` roles (if the role specified is different from `Everyone` and `Authenticated`, the result is generated by a call to `HttpContext.Current.User.IsInRole`). Also, `UserInfo` returns further user information through properties such as `IsAdministrator`, `IsAuthenticated`, `MayView`, `MayEdit`, and so on.

Finally, if the user isn't allowed to view the requested resource, they are redirected to a login page:

```
// Make sure the user can view the page
if (!objUserInfo.MayView)
    CommunityGlobals.ForceLogin();
}
```

Let's talk about the tables that store section security information. `Community_UserRoles` contains information about the existing user roles for all existing communities. `Community_Users` stores user data (all details you enter when creating a new user account) for all users in all communities. `Community_UsersInRoles` associates users with roles.

`Community_SectionSecurity` contains security information for existing community sections. In particular, it contains two important fields:

- `ss_RoleTypes` specifies the permission
- `ss_Roles` specifies the role that has access to that permission

So, one record in `Community_SectionSecurity` states something like "Community_Everyone has the View permission on the Mission section".

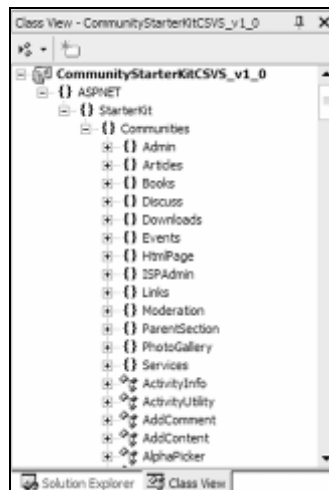
The role types are stored as integers corresponding to the values stored in the `RoleType` enumeration (`Engine/Framework/Users/Components/RoleTypes.cs`):

```
public enum RoleType {
    ViewRole = 0,
    AddRole = 1,
    EditRole = 2,
    DeleteRole = 3,
    CommentRole = 4,
    RateRole = 5,
    ModerateRole = 6
}
```

Processing the Request in the Middle Tier

The functionality in `communityDefault.aspx` is based on a large number of classes from the CSK engine. We met some of these classes earlier, and now it's time to have a closer look at them. Bear in mind that we won't attempt to look at each of them in this chapter (this would probably be, if not impossible, at least inefficient)—but we'll focus on the most important ones, giving you the knowledge you need to build your own CSK modules and features. Then, in the following chapters, we'll apply this knowledge by implementing custom functionality in the CSK.

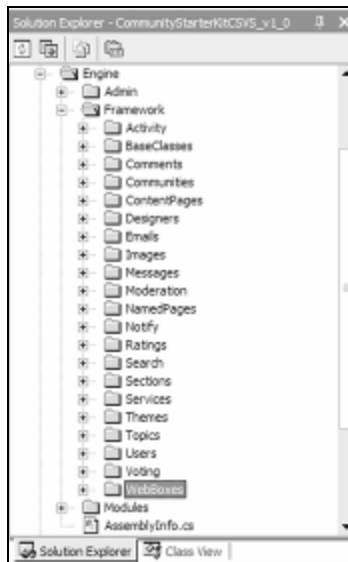
We can easily access the list of classes that compose the CSK by using the Class View window in Visual Studio .NET. All the CSK classes are located under the `ASPNET.StarterKit.Communities` namespace and each content module adds another namespace suffix to its classes:



Because these classes are arranged in a clear structure under the Engine folder, we'll use their physical location as a base for analyzing them.

The Community Starter Kit Framework

The Framework folder contains classes that support most functionality in the CSK. You can see its subfolders in the following screenshot. The self-describing folder names make it easy to guess what kind of functionality is hidden inside:

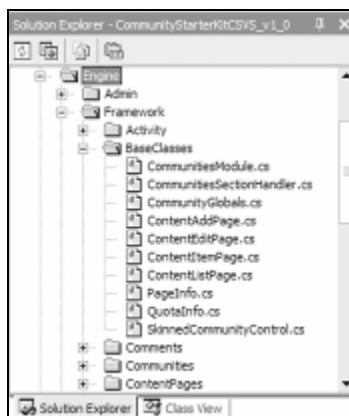


In the following pages we'll see what classes these folders contain and how they work. Many of the (sub)folders divide their classes in three additional subfolders:

- **components:** Here you can usually find utility classes that support generic functionality for the other classes in the main folder.
- **content:** The classes in this folder are custom web controls that, based on skin files, generate output for the sections or modules of the CSK.
- **controls:** Here you can find custom web controls that are used in skin files.

The Framework Base Classes

Here we'll talk about the `BaseClasses` folder, which is a very important folder because it contains classes at the core of the CSK architecture. All its classes are part of the `ASPNET.StarterKit.Communities` namespace.



We analyzed the `CommunitiesModule` class earlier. Let's see what the other classes in `BaseClasses` do.

The `CommunitiesSectionHandler` Class

`CommunitiesSectionHandler` is the configuration handler class used when defining the `communityStarterKit` group in `web.config`. We saw this class earlier when talking about the `web.config` file.

The `CommunityGlobals` Class

`CommunityGlobals` contains a number of static methods with miscellaneous functionality (such as the `GetSectionPath` and `GetPageName`), and static properties that return global data, much of it being ultimately stored in `web.config` or the database (such as `CommunityName`, `ConnectionString`, and `PrimaryDomain`).

The `SkinnedCommunityControl` Class

`SkinnedCommunityControl` is the base class for the controls in the CSK, including the four control classes we will discuss next. This class offers the required functionality to load and generate client output based on skin files. `SkinnedCommunityControl` derives from `WebControl`.

`ContentAddPage`, `ContentEditPage`, `ContentItemPage`, and `ContentListPage`

These four classes are the base classes for section-specific controls such as `AddArticle`, `EditArticle`, `Article`, or `ArticleSection`. We will look at the `Article` module in detail in Chapter 7. These controls use skin files to build content community pages, and are located in `Engine/Modules/module_name/Content` folders (where `module_name` can be `Articles`, `Books`, `Downloads`, and so on). The module classes are located in the namespace called `ASPNET.StarterKit.Communities.module_name`.

ContentAddPage is the base class for the add-content type of pages (such as the AddArticle or AddBook classes). ContentEditPage is the base class for edit-content pages (such as the EditArticle or EditBook classes). ContentListPage is the base class for pages that show lists of items (such as the ArticleSection or BookSection classes). ContentItemPage is the base class for pages that show the details of a particular content item (such as the Article or Book classes). We'll learn more about these module-specific controls later, when we'll discuss the Engine/Modules folder.

ContentAddPage, ContentEditPage, ContentItemPage, and ContentListPage inherit from SkinnedCommunityControl but provide specific functionality for the page type they are intended to represent.

It's worth taking a look at one of these to see what this means. ContentAddPage is defined like follows:

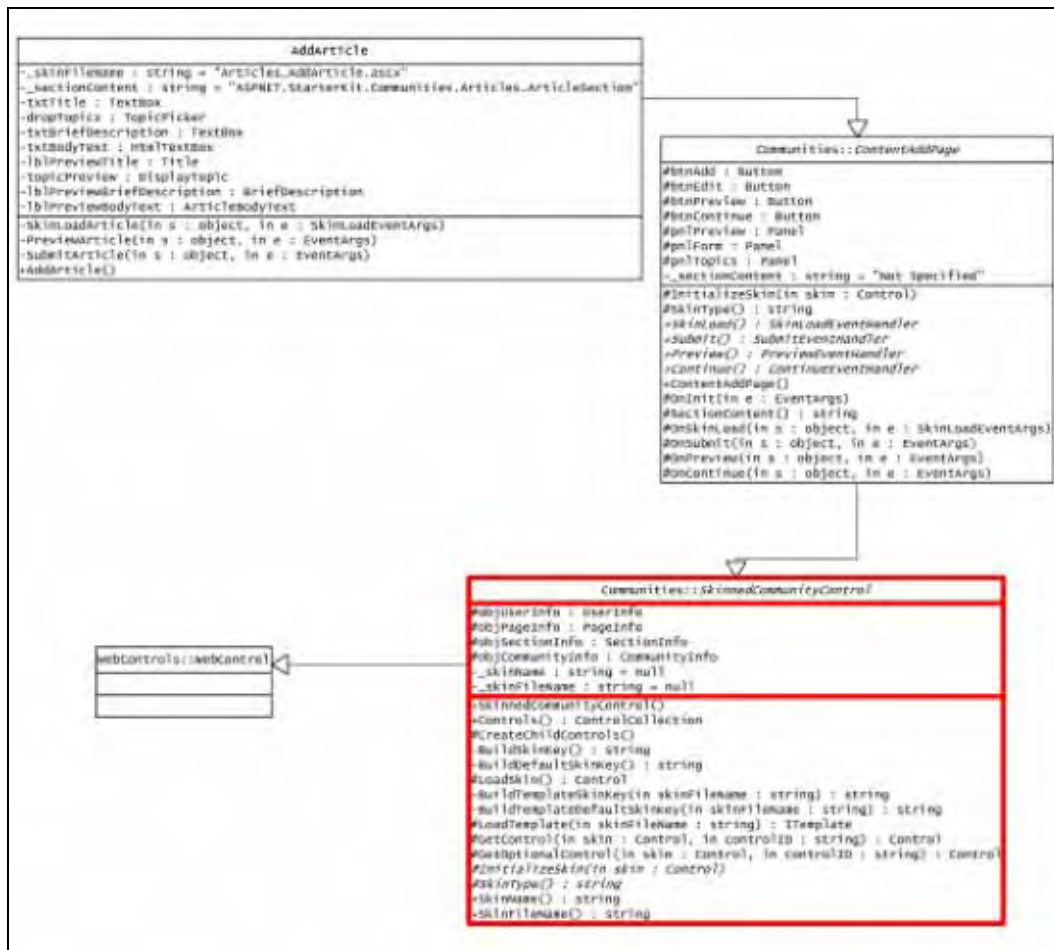
```
public abstract class ContentAddPage : SkinnedCommunityControl {  
    protected Button btnAdd;  
    protected Button btnEdit;  
    protected Button btnPreview;  
    protected Button btnContinue;  
    protected Panel pnlPreview;  
    protected Panel pnlForm;  
    protected Panel pnlTopics;  
}
```

The class defines controls that exist in any AddModule_Name pages, such as the Add and Continue buttons. Later, in the InitializeSkin method, these controls are associated with controls in the skin file:

```
// Find Add Button  
btnAdd = (Button)GetControl(skin, "btnAdd");  
btnAdd.Click += new EventHandler(OnSubmit);
```

This is the reason why you should provide a standard set of controls with predefined names when building a new skin file. The classes that inherit from ContentAddPage (such as AddArticle) add their own specific control definitions to the set and are associated with a skin to be used when rendering the output.

The following UML diagram (generated with Microsoft Visio) shows the inheritance relationship between AddArticle, ContentAddPage, SkinnedCommunityControl, and WebControl:



Similar relationships exist for the three other classes discussed earlier. The diagram reveals that **AddArticle** is the only instantiable class, since **ContentAddPage** and **SkinnedCommunityControl** are abstract classes. Note the `_skinFileName` and `_sectionContent` fields in **AddArticle**, whose values provide the information that the class needs to render its HTML output for the client. We'll discuss the **AddArticle** class with more details later, when we get to the **Engine/Modules** folder.

The PageInfo Class

PageInfo stores information about the page being requested by the visitor. This class is used in many places throughout the CSK, including `communityDefault.aspx.cs`, which uses it to build the client response.

The following code snippet shows the private members of `PageInfo`, revealing the information this object stores:

```
public class PageInfo {  
    int _id;  
    string _name;  
    int _parentID;  
    int _type;  
  
    string _title = String.Empty;  
    string _description;  
    string _metaDesc;  
    string _metaKeys;  
    string _content = String.Empty;  
}
```

The `_content` private member contains the fully qualified name of the class representing the page being loaded, such as `ASPNET.StarterKit.Communities.ParentSection.ParentSection` or `ASPNET.StarterKit.Communities.Articles.ArticleSection`.

The QuotaInfo Class

`QuotaInfo` stores quota information for the current community and current user. The four public properties of `QuotaInfo` are: `CommunityQuota`, `CommunityQuotaUsed`, `UserQuota`, and `UserQuotaUsed`.

`CommunityGlobals.GetQuotaInfo` returns a populated `QuotaInfo` object, and gets the necessary data by calling the `Community_GetQuotaInfo` stored procedure.

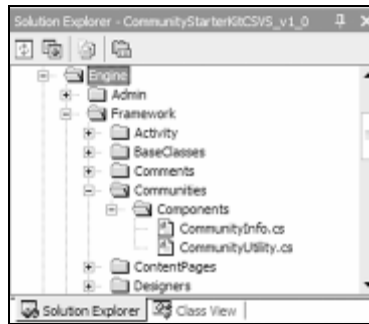
The information for all communities is stored in the same database.

`CommunityQuotaUsed` is calculated by summing the size occupied by the file downloads (remember these are stored in the database), section images, and community images.

`UserQuotaUsed` is calculated by summing the size of the file downloads and section images added by that user.

Handling Communities

The `Communities` folder contains just two classes in the `Components` folder: `CommunityInfo` and `CommunityUtility`:



CommunityInfo contains a number of private members that store community information:

```
public class CommunityInfo {
    private int _id;
    private string _name;
    private Guid _uniqueID;
    private string _primaryDomain;
    private string _domain;
    private string _subDomain;
    private string _smtpServer;
    private int _databaseQuota;
    private bool _isDisplayed;
    private bool _enableTopicMenu;
```

These private members are exposed through read-only public properties (that have only get accessors). The class can only be instantiated using its unique constructor that sets values for all these private members:

```
public CommunityInfo(DataRow draw) {
    _id = (int)draw["community_id"];
    _uniqueID = (Guid)draw["community_uniqueID"];
    _primaryDomain = (string)draw["community_primaryDomain"];
    _domain = (string)draw["community_domain"];
    _subDomain = (string)draw["community_subDomain"];
    _smtpServer = (string)draw["community_smtpServer"];
    _name = (string)draw["community_name"];
    _databaseQuota = (int)draw["community_databaseQuota"];
    _isDisplayed = (bool)draw["community_isDisplayed"];
    _enableTopicMenu = (bool)draw["community_enableTopicMenu"];
}
```

A CommunityInfo instance is built using methods from the CommunityUtility class. CommunityUtility contains only static utility methods for working with communities. The only method of this class we saw in this chapter is GetCommunityInfo, which returns a CommunityInfo object. CommunityUtility.GetCommunityInfo instantiates a CommunityInfo object with information from the database (or cache, if it exists in cache).

Apart from GetCommunityInfo, the other public static methods in CommunityUtility are called from the ISP Admin page to manipulate community information.

GetAllCommunities returns a DataSet object containing information about all existing communities. If the DataSet is not available in the cache, it is retrieved from the database using the GetAllCommunitiesFromDB private method, which gets the information from the database.

GetAllCommunitiesFromDB uses the Community_CommunitiesGetAllCommunities stored procedure to get the data.

AddCommunity, UpdateCommunity, and DeleteCommunity perform the three basic community manipulation tasks. They call the following stored procedures:

- Community_CommunitiesAddCommunity
- Community_CommunitiesUpdateCommunity
- Community_CommunitiesDeleteCommunity

GetAllCommunitiesSorted calls GetAllCommunities to get a DataSet with communities and sorts the data based on a community_sortOrder field. The position of a community in the list of communities is calculated with a ranking value stored in the community_sortOrder field of the Community_Communities table. When adding a new community, the value of community_sortOrder will be the highest value until then. When moving a community up and down in the ISP Admin menu, its ranking is switched with the community above or below it.

MoveCommunityUp and MoveCommunityDown move a community up or down in the list of communities by calling the Community_CommunitiesMoveUp or Community_CommunitiesMoveDown stored procedures.

There are four more methods used to import, export, or apply the sample content (ApplySampleContent, ApplySampleFreeContent, ExportSampleData, and ImportSampleData).

Handling Sections

The community sections are the ones you can view in Community Administration Menu | Edit Sections (such as the Articles or Downloads sections). The Sections folder contains three classes that represent and manipulate community sections: SectionInfo, SectionCollection, and SectionUtility.

We saw earlier in this chapter how the SectionInfo object was used to get and store information about sections while building the output client pages:

```
// Figure out the section and add to context
SectionInfo objSectionInfo =
    SectionUtility.GetSectionInfoFromPath(requestBasePath);
```

The SectionInfo object represents such a section and contains all the data required to describe it. SectionUtility and SectionCollection are helper classes. This data is stored internally in a number of private members. The member names are self-describing:

```
public class SectionInfo {
    int _id;
    int _communityID;
    string _name;
    string _title;
    string _menuTitle;
    string _path;
    string _description;
    bool _isEnabled;
    bool _isSystem;
    int _pageType;
    bool _isWebBoxesInherited;
    string[] _webBoxes;
```

```

int _webBoxDisplayMode;
bool _isWebServiceBoxesInherited;
string[] _webServiceBoxes;
int _webServiceBoxDisplayMode;
string _skin;
string _style;
int _parentSectionID;
string _logo;
string _pageHeader;
string _pageFooter;
string _footer;
string _pageMetaKeys;
string _pageMetaDesc;
string[] _viewRoles;
string[] _addRoles;
string[] _editRoles;
string[] _deleteRoles;
int _recordsPerPage = 10;
bool _enableTopics;
bool _enableComments;
string[] _commentRoles;
bool _enableModeration;
string[] _moderateRoles;
bool _enableRatings;
bool _enableCommentRatings;
string[] _rateRoles;
bool _enableNotifications;
bool _inheritTransformations;
string _transformations;
bool _isSectionWebService;
string _webServicePassword;
AllowHtml _allowHtmlInput;
AllowHtml _allowCommentHtmlInput;
string _content;

```

You can see the number of records per page hardcoded at 10 (the `_recordsPerPage` member). Practically, all the data you enter when adding or editing a section in Community Administration Menu can be found here. These private members are exposed through public properties.

The `SectionInfo` constructor populates the values of these private members with data from its `DataRow` and `DataRow[]` parameters:

```

public SectionInfo(DataRow drowSection, DataRow[] drowSectionSecurity,
DataRow[] drowWebBoxes, DataRow[] drowWebServiceBoxes) {

```

`SectionCollection` class represents a collection of `SectionInfo` objects. It inherits from `Hashtable` (thus making it very fast on searches), and uses the base class to do all the background work. It also contains two private members (an `ArrayList` and another `HashTable`) that are also used to store the collection of `SectionInfo` objects:

```

public class SectionCollection : Hashtable {

    ArrayList orderedSections = new ArrayList();
    Hashtable sectionsByID = new Hashtable();

```

Even though each `SectionInfo` object is stored in three different collections, only its reference is stored thrice; the `SectionInfo` object is thus stored only once in memory.

The use of the three different collections is revealed in the `Add` method, which overrides the base class's `Add` method. This method is used to add a new (key-value) pair to the `Hashtable` object.

The Add method receives two Object parameters: key, which in practice will be a string representing the virtual path (such as /CommunityStarterKit/Downloads/default.aspx) to the section, and value, which is the SectionInfo object to be added to the list. Let's see SectionCollection.Add:

```
        override public void Add(Object key, Object value) {
            SectionInfo _sectionInfo = (SectionInfo)value;
            // Don't allow duplicate values
            if (Contains(key))
                return;

            if (_sectionInfo.ParentSectionID == -1)
                homeSection = _sectionInfo;

            orderedSections.Add(_sectionInfo);
            sectionsByID.Add(_sectionInfo.ID, value);
            base.Add(key, value);
        }
```

SectionCollection.Add adds the SectionInfo object received as parameter to three different lists: to the orderedSections ArrayList, to the sectionsByID Hashtable, and to the base Hashtable object.

The indexer in SectionCollection takes a string as parameter (which represents the virtual path to the section, such as /CommunityStarterKit/Downloads/default.aspx) and returns a SectionInfo object from the base Hashtable:

```
        public SectionInfo this[string path] {
            get {
                return (SectionInfo)base[path];
            }
        }
```

The other Hashtable object is used to get a SectionInfo object given its ID, in the GetSectionByID method:

```
        public SectionInfo GetSectionByID(int sectionID) {
            return (SectionInfo)sectionsByID[sectionID];
        }
```

The orderedSections ArrayList is returned from GetOrderedSections:

```
        public ArrayList GetOrderedSections() {
            return orderedSections;
        }
```

You can see these methods called from SectionUtility, which manipulates SectionInfo objects.

The SectionUtility class contains many public methods and properties that provide the functionality required to manipulate community sections and SectionInfo objects. Let's look at the SectionUtility.GetSectionInfoFromPath method (listed in the following code snip), which we saw in action earlier in this chapter.

GetSectionInfoFromPath returns a SectionInfo object that corresponds to the virtual path received as parameter. The method first creates a SectionCollection object named sectionCollection, using the GetAllEnabledSections method of SectionUtility. Remember that the indexer of SectionCollection returns the SectionInfo object associated with the virtual path provided:

```

    public static SectionInfo GetSectionInfoFromPath(string
        requestBasePath) {
        SectionCollection sectionCollection = GetAllEnabledSections();
        return sectionCollection[requestBasePath + "default.aspx"];
    }

```

`GetAllEnabledSections` follows a common pattern in the CSK classes—it first tries to get the list of sections from the cache collection. If the list cannot be found in the cache, it is retrieved from the database (using `GetAllEnabledSectionsFromDB`), and then saved to the cache. This way, only on the first request is the database queried for data; on all subsequent requests, the fast Cache object is used instead.

```

    public static SectionCollection GetAllEnabledSections() {
        HttpContext Context = HttpContext.Current;

        SectionCollection sectionCollection =
            (SectionCollection)Context.Cache[
                CommunityGlobals.CacheKey("Sections") ];
        if (sectionCollection == null) {
            sectionCollection = GetAllEnabledSectionsFromDB();
            Context.Cache[ CommunityGlobals.CacheKey("Sections") ] =
                sectionCollection;
        }
        return sectionCollection;
    }

```

Handling Images

The Images folder contains classes that work with community images. The CSK stores part of its image files directly in the file system, while another part is stored in the database.

A rule of thumb is that static images (that you can't, for example, modify from the Community Administration Menu) are stored directly in the file system. Such examples are the skin-specific image files (the images in the `/Communities` folder), the advertisement images (which are stored in the `/Advertisements` folder), and the images in the Admin and ISPAdmin pages.

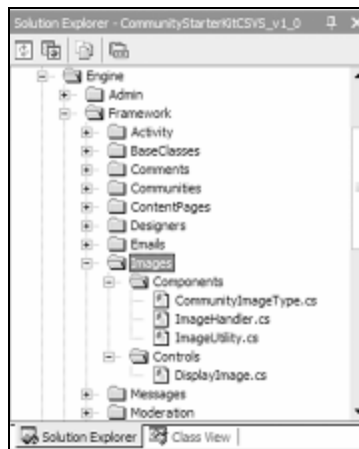
The images in the database are of two kinds: community images and section images. Community images and section images are stored in separate database tables.

Section images are the ones that appear in content pages, such as the book images, event images, or images in the Photo Gallery section. These images have numeric names, and are identified as being section images by this property.

Community images are "global" community images, which are not specific to any section. The most obvious example is the Lunar logo image. Community images don't have numerical names.

When adding a community image to the database (such as when changing the logo of your community), you must ensure the file name isn't numerical (such as `99.gif`), because the CSK will mistakenly assume it's a section image and will not be able to find it when requested from the database. Section images are automatically associated with numerical names when added to the database.

Let's get back to our Images folder now and see what it contains:



The two important classes you need to know about are `ImageHandler` and `ImageUtility`. `ImageHandler` is the class that was registered in `Web.Config` to handle image file requests by generating the output from the database. `ImageUtility` has many static methods that retrieve, add, update, or remove images from or to the database.

You'll notice that every folder that contains images (so they don't need to be read from the database) has a `Web.Config` file that unhooks image file requests from the `ImageHandler` class.

`ImageHandler` analyzes the name of the requested file and decides if it is a community image or a section image. If the name is numerical, it assumes it's a section image and calls `ImageUtility.GetSectionImage`, which retrieves the image (for this, it uses the `Community_ImagesGetSectionImage` stored procedure).

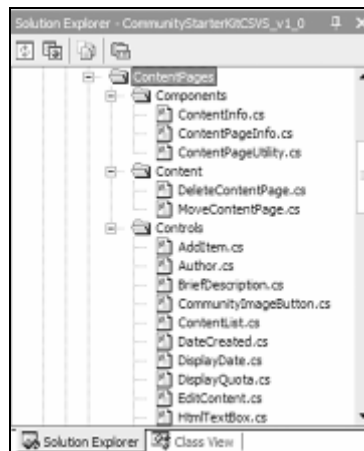
If the name is not numerical, it calls `ImageUtility.GetCommunityImage`, which in turn calls the `Community_ImagesGetCommunityImage` stored procedure.

Inside the database, community images are stored in the `Community_CommunityImages` table and section images in the `Community_SectionImages` table.

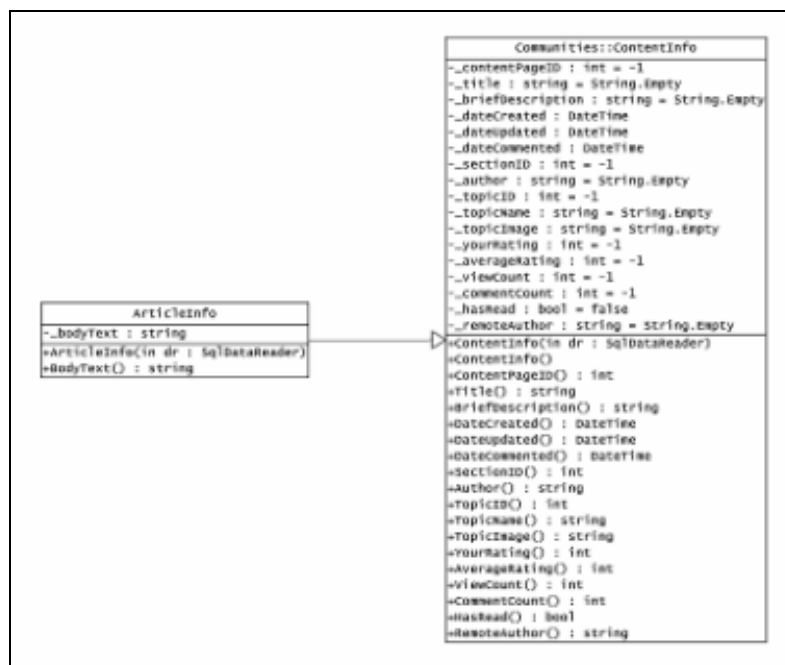
`DisplayImage`, the single control in the `Controls` class, is the base class for the controls that display section images, such as `DisplayBookImage`, `DisplayEventImage`, or `DisplayPhotoImage`. It generates the `img` HTML tags for database-generated images.

Handling Content Pages

The `ContentPages` folder contains the basic functionality required to build content pages. Its structure is shown in the following screenshot:



The Components folder contains three classes: ContentInfo, ContentPageInfo, and ContentPageUtility. ContentInfo represents all the information about a particular content item (such a book or an article) and is the base class for all ModuleInfo-type classes (like ArticleInfo, BookInfo, and so on). The following UML diagram shows the inheritance relationship between ArticleInfo and ContentInfo:



You can see the private and public members of ContentInfo, whose names are self-describing and reveal what kind of information it stores.

ContentPageInfo is also used when building content pages. ContentPageInfo stores information about the web page displaying a content item, such as the page title and description. The following listing shows the private members of ContentPageInfo:

```
public class ContentPageInfo {  
    int _contentPageID;  
    int _parentID;  
    int _sectionID;  
    int _pageType;  
    string _title;  
    string _description;  
    string _metaDescription;  
    string _metaKeys;  
    string _pageContent;  
    ModerationStatus _moderationStatus;  
}
```

ContentPageUtility is the utility class that manipulates content pages.

The Content folder in ContentPages contains two controls: DeleteContentPage (which builds its output based on ContentPages_DeleteContentPage.ascx) and MoveContentPage (ContentPages_MoveContentPage.ascx).

The Controls folder contains a large number of controls that are used in many CSK skin files.

Handling Search

The Community Starter Kit implements a custom search feature instead of relying on SQL Server's full-text search, making it possible to use MSDE as the database server.

The typical search method when full-text is not available is to split the search string into words, and then look into the database for these words. However, the search algorithm in the Community Starter Kit goes the other way round: it takes every word from the searchable database content and verifies if it exists in the search string. This method doesn't offer much flexibility, but it works.

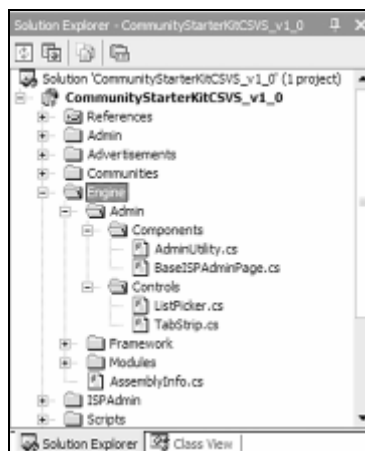
The searchable content is composed of the content item's title and brief description. These are split into separate words named search keys. The search keys are saved into a special data table (Community_SearchKeys) whenever new content is added or updated into the database.

The Search folder contains two classes: a utility class named SearchUtility, and a control named Search that represents Search_Search.ascx. SearchUtility has public static methods, which allow editing and adding search keys to the database, and a method named GetSearchResults that performs the actual searching.

Apart from GetSearchResults, the other methods in SearchUtility are used in content-specific utility classes, such as ArticleUtility, BookUtility, and so on. These need to update the database search keys whenever the community content changes.

Studying the Engine/Admin Folder

Engine/Admin supports the Admin and ISPAAdmin pages. It has two subfolders: Components and Controls, each of them having two files as shown in the following screenshot:



In Components we find two classes: `AdminUtility` and `BaseISPAdminPage`. `AdminUtility` supports the actions in the Community Administration Menu, such as editing sections, and so on. `BaseISPAdminPage` represents the base page from which all ISP Admin pages derive, and its role is to handle security for these pages (it redirects to `IspLogin.aspx` if the ISP Admin isn't authenticated).

`AdminUtility` and the two custom controls are placed in the `ASPNET.StarterKit.Communities.Admin` namespace. `BaseISPAdminPage` is part of `ASPNET.StarterKit.Communities`.

The Controls folder contains two custom controls `ListPicker` and `TabStrip` that are used in various Community Administration Menu pages.

Modules for the CSK

Here we discuss the `Engine/Modules` folder. A CSK module represents a specific content type, such as articles or books. The `Modules` folder contains the functionality that supports the various CSK modules.

Each subfolder in `Modules` represents one module type. The subfolders are: `Articles`, `Books`, `CustomModules`, `Discuss`, `Downloads`, `Events`, `HtmlPage`, `Links`, `ParentSection`, `PhotoGallery`. `ParentSection` represents CSK parent sections—in the Lunar .NET community website there are two parent sections, `Home` and `Discuss`.

All the classes inside a module are placed in a namespace named after the module. For example, the classes in the `Articles` folder are located in the `ASPNET.StarterKit.Communities.Articles` namespace.

Every module folder has the same structure formed of three subfolders: `Components`, `Content`, and `Controls`.

The following screenshot shows the `Modules` folder and the contents of the `Articles` subfolder:



The `Components` folder contains classes that provide generic functionality for the other classes in the `Articles` module. `ArticleInfo` represents data for a specific article, and `ArticleUtility` offers the means to manipulate articles in the database, such as to add, update, or delete articles, or to retrieve article data in form of `ArticleInfo` objects. We learned earlier that `ArticleInfo` inherits from `ContentInfo`, which is the base class for all `ModuleInfo`-type classes.

The classes in the `Content` folder are the custom controls that generate the content section of an article community page. When discussing `communityDefault.aspx`, we saw how these classes are instantiated with `Activator.CreateInstance` based on their fully qualified name.

In `Articles/Content` we find four classes:

- `AddArticle`: Represents the page that shows up to add a new article
- `Article`: Represents the article details page
- `ArticleSection`: Represents the page showing the complete list of articles
- `EditArticle`: Represents the page that edits an article

These classes generate their output by using skin files and have similar structure.

Finally, the classes in the `Controls` folder represent custom controls used in `Articles` skins. Unlike the controls in `Engine/Framework/ContentPages/Controls`, which are intended to be use in various skin files, the `Controls` folder of a module contains controls specific to that module. In our case, `ArticleBodyText` represents the body text of an article, and `ArticleEditContent` the add, edit, delete, move, comment, and moderate links for an article.

Other modules (such as the Books module) have much more specific custom controls. Remember that controls that start with Item are intended to be used inside ContentListXXXX controls. In Chapter 7 we'll look at a CSK module in detail.

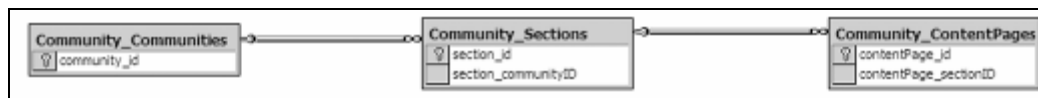
The Database Structure

The database contains a large number of data tables, which are always accessed and manipulated through stored procedures. In the following pages, we'll have a quick look at the way the CommunityStarterKit database is organized. In Chapter 7, we'll learn about the stored procedures used to manipulate module data.

The Tables

Remember that the name of the database can be specified when installing the Community Starter Kit. Here we'll just assume that you used the default name, CommunityStarterKit.

The CommunityStarterKit database has quite a complicated structure, formed of more than 60 data tables (the names of which always start with Community), but its skeleton is composed of just three important tables: Community_Communities, Community_Sections, and Community_ContentPages. These tables have one-to-many relationships enforced with foreign keys as shown in the following figure:

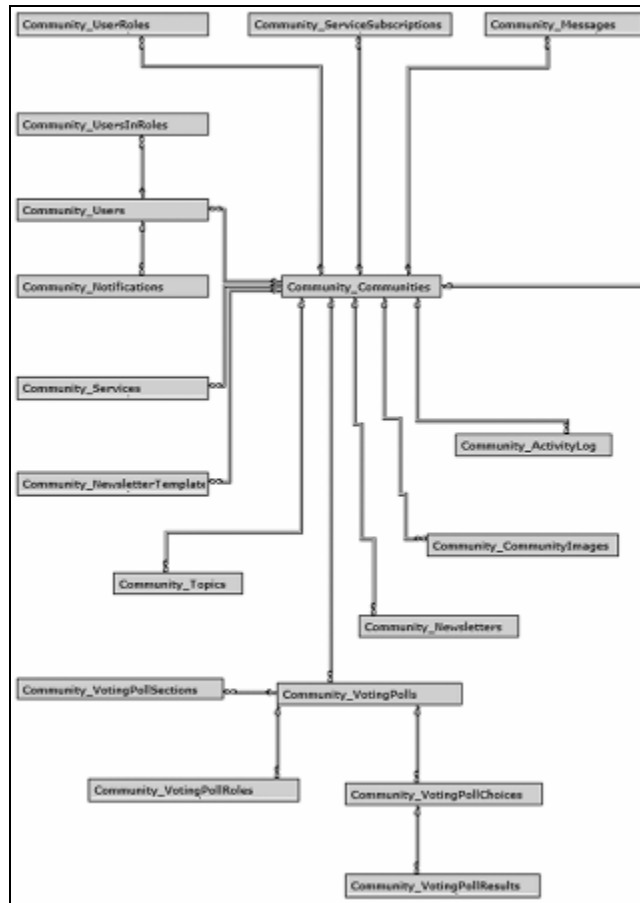


Community_Communities contains information about each community in the CSK installation. Basically, it contains the data you enter when creating a new community in the ISP Admin page, which gets stored in CommunityInfo objects:

	Column Name	Data Type	Length	Allow Nulls
?	community_id	int	4	
	community_name	nvarchar	50	
	community_uniqueID	uniqueidentifier	16	
	community_PrimaryDomain	nvarchar	50	
	community_domain	nvarchar	50	
	community_subdomain	nvarchar	50	
	community_smtpServer	nvarchar	50	
	community_databaseQuota	int	4	
	community_databaseQuotaUsed	int	4	
	community_sortOrder	int	4	
	community_isDisplayed	bit	1	
	community_enableTopicMenu	bit	1	

There are a number of other tables that relate to Community_Communities and contain information related to the community as a whole, and not to particular community sections or modules.

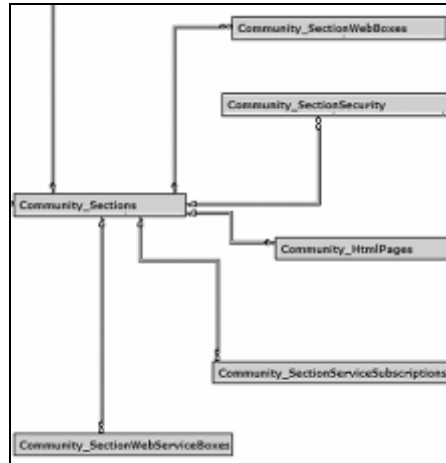
The following figure shows these tables:



As shown in the preceding figure, `Community_Communities` also relates to `Community_Sections`, which stores the sections that exist in that community (and also for all the other communities). These are the sections that you can see in edit sections in the Community Administration Menu. The structure of `Community_Sections` is as follows:

Community_Sections				
	Column Name	Data Type	Length	Allow Nulls
✓	section_id	int	4	
	section_communityID	int	4	
	section_name	nvarchar	50	
	section_title	nvarchar	100	
	section_menuTitle	nvarchar	50	
	section_description	nvarchar	500	
	section_isEnabled	bit	1	
	section_pageType	int	4	
	section_isWebBoxesInheri	bit	1	
	section_isWebboxDisplayMo	int	4	
	section_isWebServiceBoxe	bit	1	
	section_isWebServiceBoxDir	int	4	

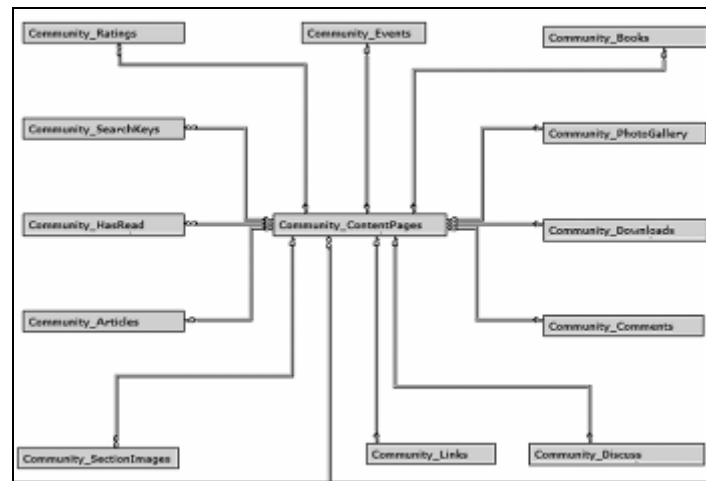
The tables that relate to `Community_Sections` contain other section-related data:



Finally, each record in `Community_Sections` relates to one or more content pages, which are stored in `Community_ContentPages`:

Column Name	Data Type	Length	Allow Nulls
contentPage_id	int	4	
contentPage_communityID	int	4	
contentPage_userID	int	4	✓
contentPage_editUserID	int	4	✓
contentPage_parentID	int	4	
contentPage_pageType	int	4	
contentPage_sectionID	int	4	
contentPage_title	nvarchar	100	
contentPage_description	nvarchar	500	
contentPage_metaDesc	nvarchar	250	
contentPage_metaKeys	nvarchar	250	
contentPage_dateCreated	datetime	8	
contentPage_dateUpdated	datetime	8	✓
contentPage_dateCommented	datetime	8	✓
contentPage_topicID	int	4	✓
contentPage_replyID	int	4	
contentPage_ModerationStatus	int	4	
contentPage_ModerationUserID	int	4	
contentPage_Rating	int	4	
contentPage_SortOrder	datetime	8	
contentPage_dateVisible	datetime	8	
contentPage_viewCount	int	4	
contentPage_timestamp	timestamp	8	
contentPage_remoteCommunityUniqueID	uniqueidentifier	16	✓
contentPage_remoteContentPageID	int	4	✓
contentPage_remoteUsername	nvarchar	50	✓
contentPage_remoteTimestamp	varbinary	50	✓

Each record in `Community_ContentPages` contains details about each content item in a section. The actual data is stored in separate tables, which relate to `Community_ContentPages` and which are tailored to specific types of content (such as Articles, Books, and so on):



It's worth knowing that the CommunityStarterKit database also contains a number of tables that contain the sample CSK data for the Lunar .NET community. For example, there is a table named Community_SampleArticles that is used to populate the Community_Articles table with sample data when installing the Lunar .NET community.

Online CSK Resources

Because of the popularity of the Community Starter Kit, you can find a number of modules and updates on the Internet. Apart from the very useful topics you can find on the CSK ASP.NET forums, here are a few other useful resources:

- <http://www.reddnet.com/>: Stephen M. Redd's hobby site, web-log, and personal home page
- <http://www.colin-munford.me.uk/>: Colin Munford's Community Site
- <http://csk.matthewroche.com>: Matthew Roche's ASP.NET Community Starter Kit resource site
- <http://www.learnmsnet.com/Articles/SKArticles/default.aspx>: Various articles about the Community Starter Kit
- <http://www.daveranck.com/>: Personal website of Dave Ranck (Colin and Dave were also technical reviewers on this book.)

Summary

Here we are, at the end of a long chapter on the core of the CSK architecture. We learned that the CSK builds client output using the CommunitiesModule class and the communityDefault.aspx web form. We saw how the important classes are structured in the CSK project's folders and what they do. Finally, we saw the organization of the CSK database.

7

Inside a CSK Module

In this chapter, we will take a look at how the modules in the CSK work. Specifically, we will walk through all areas of the Articles module from database schema to presentation layer and gain an understanding of the module architecture. This understanding will help you customize the CSK, and will also lay the foundation for Chapter 8 where you build a new module from scratch.

Conceptually, module architecture is similar to the following diagram:

	Articles	Books	Downloads	
Moderation				
Sorting				
Search				
Ratings				
Topics				
Comments				
Web Service				
Skinning				
	Content			

Each module builds upon a **content foundation**. This foundation uses common database tables, SELECT statements, C# base classes, and user controls for display. Each module extends and adds to these common abstractions by adding additional tables, stored procedures, and user controls.

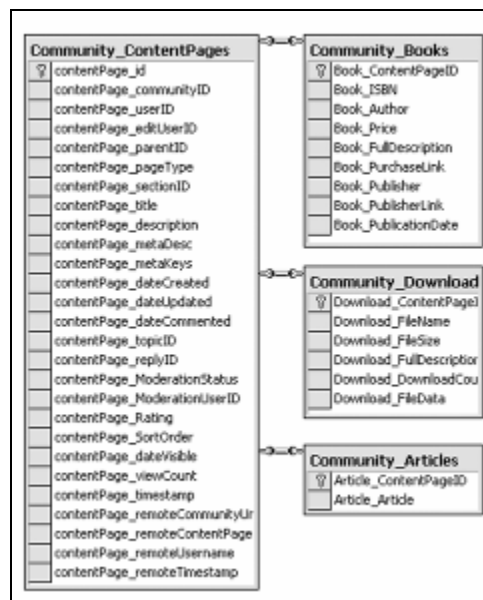
Content represents the common foundation that allows the CSK to weave functionality across modules. For example, all of the modules in the CSK allow users to sort, search, and rate content. You can comment on any item in any module, and you can assign distinct topics to items in each

module. Even though each module builds upon the content foundation in varying ways for different purposes, each module also allows consistent behavior by building on common ground.

All of the modules use the same patterns and paradigms, so having a detailed understanding of just one of the CSK modules will give you enough information to find your way through any of the other modules.

Database Layer

The center of the universe for the CSK is the `Community_ContentPages` table. This table contains all the important attributes for nearly anything that can be displayed in the CSK. As you can see in the following figure, the table can track titles, descriptions, ratings, view counts, and more. All of the attributes common to every module live here, and from this table the CSK builds the cross-module functionality we discussed in the previous diagram.



All of the modules start by placing an entry in the `Community_ContentPages` table. For example, a book's title is kept in this table, as is the title for downloads and articles.

If a module has additional attributes to store, it can then create a second table linked to `Community_ContentPages` by `contentPage_id`. The `Community_Books` table stores attributes specific to a book, such as the ISBN and book author name, while the `Community_Articles` table keeps the complete text of all the articles in the system. Each of these module-specific tables builds on the `Community_ContentPages` table instead of storing attributes such as title and description inside itself.

The advantages of this design become apparent when we look at the stored procedures and user-defined functions associated with content pages.

Procedures and Functions

All data access to the community tables happens through stored procedures. In each module you will generally find a set of four stored procedures to add content, edit content, retrieve a single piece of content, and retrieve a sorted list of content. For the Articles module, the stored procedures are:

- Community_ArticlesAddArticle
- Community_ArticlesEditArticle
- Community_ArticlesGetArticle
- Community_ArticlesGetArticles

You might notice the lack of a stored procedure to delete content. However, the CSK offers this functionality. Deletion is a special case we will revisit later in this chapter.

There are two additional stored procedures provided by each module for a remote site to retrieve and update content. These procedures are `Community_ArticleServiceSelect` and `Community_ArticleServiceUpdate`. These two procedures are extremely similar to the `Select` and `Update` stored procedures but require slightly different parameters and return modified result sets. For each module, you'll generally see a total of six stored procedures for access to the content. Let's first drill down into the code to add an article.

Community_ArticlesAddArticle

We will look at the stored procedure used to add an article to the list of content. The procedure first performs some simple parameter mapping. For example, we pass username as a parameter from the ASP.NET layer, as the username is readily available; however, the content table needs to store a user ID, which is a foreign key in the `Community_Users` table. The CSK provides a user-defined function (`Community_GetUserID`), which is used by the procedure to look up the user ID for a given community ID and username. Another parameter mapping UDF is `Community_GetPageTypeFromName`, which takes the string representation of a page type and returns a `pageTypeID` from the `Community_PageTypes` table.

```
CREATE PROCEDURE Community_ArticlesAddArticle
(
    @communityID int,
    @sectionID int,
    @username NVarchar(50),
    @topicID int,
    @title NVarchar(100),
    @introText NVarchar(500),
    @metaDescription NVarchar(250),
    @metaKeys NVarchar(250),
    @moderationStatus int,
    @bodyText Text
)
AS
DECLARE @ContentPageID int
-- Get Page Type
DECLARE @pageType int
SET @pageType = dbo.Community_GetPageTypeFromName('Article')
-- Get User ID
DECLARE @userID int
SET @userID = dbo.Community_GetUserID(@communityID, @username);
-- Add content page
```

```

BEGIN TRAN
EXEC @ContentPageID = Community_AddContentPage
    @communityID,
    @sectionID,
    @userID,
    @title,
    @introText,
    @metaDescription,
    @metaKeys,
    @pageType,
    @moderationStatus,
    @topicID
INSERT Community_Articles
(
    Article_ContentPageID,
    Article_Article
)
VALUES
(
    @ContentPageID,
    @bodyText
)
COMMIT TRAN
RETURN @ContentPageID

```

This procedure relies on a second procedure (`Community_AddContentPage`) to do most of the work. `Community_AddContentPage` adds the common content attributes to the `Community_ContentPages` table using a simple `INSERT` statement with the parameters we pass. The procedure then retrieves the ID of the new record and returns the value.

`Community_ArticlesAddArticle` uses the return value in a second simple `INSERT` statement to place the article text (`Article_Article` column) into the `Community_Articles` table.

Surrounding the call to `Community_AddContentPage` and the `INSERT` statement is a transaction. In the event of an error, the transaction will roll back changes and prevent orphaned records from appearing in the tables. On success, the procedure will commit the transaction and return the new content ID to the caller.

Community_ArticlesGetArticles

When a user first enters an article section, the CSK displays a list of available articles. The list may be paged, and the user may select various sort orders for the list (by date, by popularity, and so on). This functionality is true of any module with multiple content items, such as articles, books, and downloads.

Since paging and sorting is common functionality, the CSK provides a couple of user-defined functions to perform content paging and sorting work. These two UDFs are named `Community_GetOrderedContent` and `Community_GetPagedSortedContent`. Any module can build on top of these two functions, which will perform most of the difficult work.

Ordering Content

`Community_GetOrderedContent` begins the content processing by populating a table variable with the content IDs in sorted order. A table variable works in similar fashion to a temporary table in SQL Server, and with UDFs you can use a variable of type table as a return value from the function.

The table variable in `Community_GetOrderedContent` includes an identity field (`IndexID`), which will automatically assign increasing numbers to inserted records. The following listing is an excerpt from the function:

```
ALTER FUNCTION Community_GetOrderedContent
(
    @sectionID INT,
    @currentDate DATETIME,
    @sortOrder NVARCHAR(50),
    @includeInvisible BIT = 0
)
RETURNS @ContentTable Table
(
    IndexID INT IDENTITY (1, 1) NOT NULL,
    PageID INT
)
AS
BEGIN
    /* Sort by DateCreated ASC */
    IF @sortOrder = 'DateCreated ASC'
    BEGIN
        INSERT @ContentTable (PageID)
        SELECT
            ContentPage_ID
        FROM
            Community_ContentPages
        WHERE
            ContentPage_SectionID = @sectionID
            AND ContentPage_ParentID = -1
            AND ContentPage_ModerationStatus = 1
            AND (ContentPage_dateVisible < @currentDate OR
                @includeInvisible=1)
        ORDER BY
            ContentPage_DateCreated ASC
    RETURN
    END
    /* continues ... */
```

First, notice the `@sortOrder` parameter. When the function inspects the parameter to determine how to order the content, it compares `@sortOrder` to hardcoded strings. If you need to add a custom new sort order, you'll need to modify this function and add an additional `IF END` section. Although not shown above, the function has checks for all known sort types (`DateCreated ASC`, `DateCreated DESC`, `ViewCount ASC`, `ViewCount DESC`, and others).

The function does not simply append the `@sortOrder` parameter to a SQL statement but instead has a unique `SELECT` statement for each possible sort order. Although this technique is more verbose than using dynamic SQL, it will generally take better advantage of SQL query optimizations and lessen the risk of a SQL injection attack.

By selecting `ContentPage_ID` (with the appropriate `ORDER BY` clause) into a table with an `IDENTITY` field, the function builds a numbered list of content. For example, when the sort order is `Rating ASC`, the ID for the highest rated content piece in this section will appear in the result table with an `IndexID` of one, the second highest with an `IndexID` of two, and so on. The next function in the chain, `Community_GetPagedSortedContent`, can then select the required content items to display for a given page and page size by simply looking at the `IndexID`.

Paging and Sorting Content

The `Community_GetPagedSortedContent` UDF has two responsibilities. The first is to calculate the range of `IndexID` values to return for display. For example, on page one of a list with 10 items to a page, you would want records with an `IndexID` between 1 and 10. For the second page, the `IndexID` range is 11 to 20, and so on. Given the page size, and page index as parameters, the function uses the following code to determine the proper `IndexID` values to use:

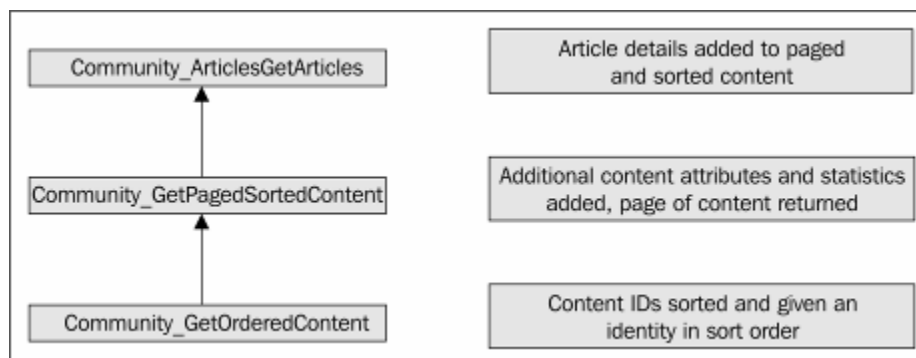
```
DECLARE @PageLowerBound int
DECLARE @PageUpperBound int
SET @PageLowerBound = @pageSize * (@pageIndex - 1)
SET @PageUpperBound = @PageLowerBound + @pageSize + 1
```

The next step is to apply these values against the ordered content. The following is an outline of the SQL statement used, with the details omitted:

```
SELECT
    IndexID,
    /* additional select list */
FROM
    dbo.Community_GetOrderedContent(@sectionID,
                                    @currentDate,
                                    @sortOrder,
                                    @includeInvisible)
    JOIN Community_ContentPages (nolock)
      ON PageID = ContentPage_ID
    /* additional join list*/
WHERE
    IndexID > @PageLowerBound AND
    IndexID < @PageUpperBound
```

This `SELECT` statement uses the calculated page bounds in the `WHERE` clause to fetch just the content needed for the current page. `Community_GetPagedSortedContent` also returns other content attributes, such as the title and description, and uses a number of other user-defined functions to grab additional information for the resultset. For instance, `Community_GetCommentCount` will return the number of user comments on each content page, while `Community_GetHasRead` determines if the current user has previously viewed a content page, returning 1 if the user has, and 0 if not.

The return value from `Community_GetPagedSortedContent` will give you everything you need to know about content from the `Community_ContentPages` table. The return value is a table variable, which is designed to allow other stored procedures and queries to build on top:



The stored procedure to return a list of articles will use `Community_GetPagesSortedContent` in this manner to perform all of the calculations and filtering necessary for paging and sorting. Almost all of the hard work has been completed by the two CSK user-defined functions, so all that is left for this final stored procedure is to add article-specific columns to the output. The entire stored procedure to return an article list is:

```
ALTER PROCEDURE Community_ArticlesGetArticles
(
    @communityID int,
    @username NVarchar(50),
    @sectionID int,
    @pageSize int,
    @pageIndex int,
    @sortOrder NVarchar(50)
)
AS
DECLARE @currentDate DATETIME
SET @currentDate = GetUtcDate()
-- Get the Articles
SELECT
    null Article_Article,
    Content.*
FROM
    dbo.Community_GetPagedSortedContent
    (
        @communityID,
        @username,
        @sectionID,
        @currentDate,
        @sortOrder,
        @pageSize,
        @pageIndex,
        default
    ) Content
ORDER BY
    IndexID
```

This procedure adds an additional column (`Article_Article`) to the resultset. This column represents the full text of an article. You'll notice, however, the stored procedure always populates this column with a `NULL` value. There are two related reasons for this:

- A list of articles would never display the full text of an article but provide just enough information for the user to decide if they want to view the full article. When the user clicks into the article, the CSK will retrieve the full text for display. By returning `NULL` values at this level the stored procedure removes information we would never display in the list.
- The CSK uses a C# class to represent an article (we will see this later in the chapter). The CSK uses this class when fetching an entire list of articles or just a single article with all the text, and the class always expects to find a column with the name `Article_Article`.

Nearly all of the modules follow a similar pattern. For instance, the stored procedure `Community_BooksGetBooks` uses `Community_GetPagedSortedContent` to retrieve content information, and then adds the author and price columns (in addition to others). The procedure also adds the `Book_FullDescription` column but places `NULL` values in the record for this column.

The CSK can then display the price and author name in a list of books, but not the full description, which is reserved for the detail page. Nevertheless, the CSK can use the same `BookInfo` object to represent a book both in a list of books and for an individual book.

Community_ArticlesGetArticle

`Community_ArticlesGetArticle` retrieves a single article from the database. This stored procedure also uses a helper function, `Community_GetContentItem`, to first retrieve the content. The helper function returns all of the needed data about a content item from `Community_ContentPages`, including derived information such as the comment count and average rating. Again, `Community_ArticlesGetArticle` will build on the resultset by adding article-specific columns, in this case a fully populated `Article_Article` column:

```
ALTER PROCEDURE Community_ArticlesGetArticle
(
    @communityID INT,
    @username NVarchar(50),
    @contentPageID int
)
AS
-- Get the UserID
DECLARE @userID INT
SET @userID = dbo.Community_GetUserID(@communityID,
    @username)
-- Update ViewCount and HasRead Stats
EXEC Community_ContentPagesTrackStats @userID,
    @contentPageID
-- We need a static date for our UDF
DECLARE @currentDate DATETIME
SET @currentDate = GetUtcDate()
-- Get Individual Article (We do a select * to make this -- procedure less
fragile)
SELECT
    Article_Article,
    Content.*
FROM
    dbo.Community_GetContentItem(@communityID,
        @userID,
        @currentDate) Content
    JOIN Community_Articles (nolock)
        ON ContentPage_ID = Article_ContentPageID
WHERE
    ContentPage_ID = @contentPageID
```

This stored procedure has one additional job—to update the view count and "has read" attributes for the article. Although this would seem to be common to all content in the CSK, the CSK designers decided not to push this functionality into the `Community_GetContentItem` function.

If you are building your own module, remember to update attributes like view count by calling `Community_ContentPagesTrackStats` from your stored procedure.

Delete Content

The CSK also allows authorized users to delete content from a community. All modules use the same stored procedure, `Community_ContentPagesDeleteContentPage` to delete content:

```
CREATE PROCEDURE Community_ContentPagesDeleteContentPage
(
    @communityID int,
    @sectionID int,
    @contentPageID int
)
AS
DELETE
    Community_ContentPages
WHERE
    ContentPage_CommunityID = @communityID
    AND ContentPage_SectionID = @sectionID
    AND (ContentPage_ID = @contentPageID OR
        ContentPage_ParentID=@contentPageID)
```

This procedure deletes content from the `Community_ContentPages` table, which is a good start, but what if we are deleting an article? How will the record in `Community_Articles` be deleted? The answer lies in how the CSK created the `Community_Articles` table:

```
CREATE TABLE [Community_Articles] (
    [Article_ContentPageID] [int] NOT NULL ,
    [Article_Article] [ntext] COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
    CONSTRAINT [PK_Community_Articles] PRIMARY KEY CLUSTERED
(
    [Article_ContentPageID]
) ON [PRIMARY] ,
    CONSTRAINT [FK_Community_Articles_Community_ContentPages] FOREIGN KEY
(
    [Article_ContentPageID]
) REFERENCES [Community_ContentPages] (
    [contentPage_id]
) ON DELETE CASCADE
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

The `ON DELETE CASCADE` defines a cascading referential action to take place. Specifically, when the primary key (`contentPage_id` in `Community_ContentPages`) is deleted, SQL Server will delete all foreign keys referencing the entry from other tables. Cascading referential integrity allows `DELETE` and `UPDATE` statements that reference key values to propagate (cascade) through all the related tables. Because of this, no module has a stored procedure to delete content but instead depends on cascading deletes.

The database is a good demonstration for the module architecture in general. The CSK builds functionality on top of common pieces by using a common table and generic functions.

Module Classes

As you will recall from Chapter 6, the CSK divides the middle layer of each module into three distinct categories:

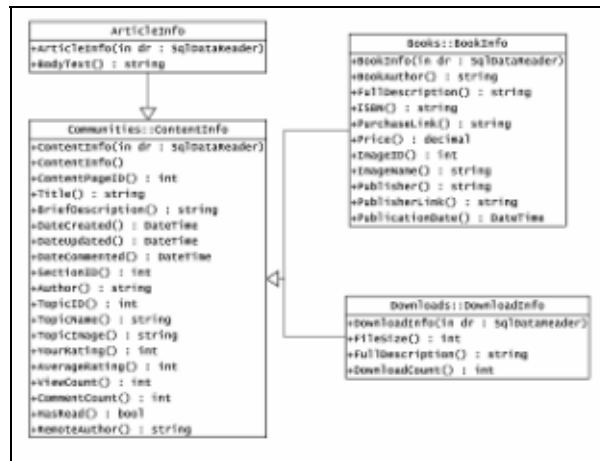
- **Components:** Contain classes used by the CSK for database access and the classes used to represent module content (that is an article or a book).

- **Content:** Includes the code-behind classes, which the CSK will attach to a skin for presentation. These classes move the information from component classes to the screen (when retrieving content), and from the screen into the component classes (when adding and editing content).
- **Controls:** Contain classes representing web controls, which render content attributes (title, description, date posted) in a web form.

The CSK solution file keeps each category in a separate directory. For articles, the directories are Engine\Modules\Articles\Components, Engine\Modules\Articles\Content, and \Engine\Modules\Articles\Controls.

Components

The following diagram is very similar to the database schema shown in the beginning of the chapter except this diagram is a UML representation of several component type classes from three different modules. Each class contains properties exposing content information, such as `Title` and `BriefDescription`, and you can map each of these properties back to a column in the resultset produced by the "get" stored procedures for the module.



The `ContentInfo` class can hold all of the fields from the `Community_GetPagedSortedContent` UDF. Just as each module's stored procedure can build on the resultset from `Community_GetPagedSortedContent`, each module's components build on top of `ContentInfo` through inheritance. So, an `ArticleInfo` object exposes all of the public properties of `ContentInfo`.

Each component class also provides a constructor with a `SqlDataReader` parameter. As shown in the following code, the constructor will pull fields out of the `SqlDataReader` and populate the object's fields:

```
public ArticleInfo(SqlDataReader dr) : base(dr)
{
```

```
        if (dr["Article_Article"] != DBNull.Value)
            _bodyText = (string)dr["Article_Article"];
    }
```

The constructor for an `ArticleInfo` object will call the base class constructor (`ContentInfo`) and pass along the `SqlDataReader` instance. This allows the `ContentInfo` object to copy fields it requires from the record. Also, `ArticleInfo` will check if the `Article_Article` column contains a `NULL` value before attempting to copy the value into a field. Remember, the `Article_Article` column always contains `NULL` values when retrieving a list of articles, but contains the full text of the article when retrieving a single article. Finally, the article constructor maps database columns to friendly names, so the `Article_Article` column becomes a string exposed by the `BodyText` property:

```
    string _bodyText;
    public string BodyText
    {
        get {return _bodyText;}
        set {_bodyText = value;}
    }
```

These info classes (`ArticleInfo`, `BookInfo`, and so on) are simple data containers that read the fields from a database record and expose the fields as properties. The utility classes (`ArticleUtility`, `BookUtility`) invoke stored procedures, create the info classes, add new content, and edit existing content. Each module utility class in the CSK provides four methods: one to add content, one to edit content, one to retrieve a list of content, and one to retrieve a single piece of content. For the `ArticleUtility` class, these methods are `AddArticle`, `EditArticle`, `GetArticles`, and `GetArticleInfo`.

The add and edit methods are similar. The `AddArticle` method is as follows:

```
    public static int AddArticle
    (
        string username,
        int sectionID,
        int topicID,
        string title,
        string introText,
        int moderationStatus,
        string bodyText
    )
    {
        SqlConnection conPortal = new
            SqlConnection(CommunityGlobals.ConnectionString);
        SqlCommand cmdAdd = new SqlCommand(
            "Community_ArticlesAddArticle", conPortal);
        cmdAdd.CommandType = CommandType.StoredProcedure;

        cmdAdd.Parameters.Add("@RETURN_VALUE",
            SqlDbType.Int).Direction =
            ParameterDirection.ReturnValue;
        cmdAdd.Parameters.Add("@communityID",
            CommunityGlobals.CommunityID);
        cmdAdd.Parameters.Add("@sectionID", sectionID);
        cmdAdd.Parameters.Add("@username", username);
        cmdAdd.Parameters.Add("@topicID", topicID);
        cmdAdd.Parameters.Add("@title", title);
        cmdAdd.Parameters.Add("@introText", introText);
        cmdAdd.Parameters.Add("@metaDescription",
            ContentPageUtility.CalculateMetaDescription(
                introText));
    }
```

```

cmdAdd.Parameters.Add("@metaKeys",
    ContentPageUtility.CalculateMetaKeys(introText));
cmdAdd.Parameters.Add("@moderationStatus",
    moderationStatus );
cmdAdd.Parameters.Add("@bodyText", SqlDbType.NText);
cmdAdd.Parameters[ "@bodyText" ].Value = bodyText;

conPortal.Open();
cmdAdd.ExecuteNonQuery();
int result =
    (int)cmdAdd.Parameters["@RETURN_VALUE"].Value;

// Add Search Keys
SearchUtility.AddSearchKeys(conPortal, sectionID,
    result, title, introText);
conPortal.Close();
return result;
}

```

Component Helpers

Both add and edit methods open a database connection and build the required parameters for the underlying stored procedure. The code also uses some helper methods in other classes. You'll find these classes useful when building your own module.

CalculateMetaDescription will truncate the incoming string at 250 characters, HTML encode the string, and return the result. CalculateMetaKeys truncates the string at 200 characters, HTML encodes the string, and then replaces all the whitespace between words with a comma. The `communityDefault.aspx` page uses these two results to create meta tags for each page of content appearing in the CSK. For example, if we have an article where the description text reads `Code To Predict Cattle Futures`, then the page will have the following meta tags sent to the client to help improve search engine indexing:

```

<META NAME="KEYWORDS" CONTENT="Code,To,Predict,Cattle,Futures"/>
<META NAME="DESCRIPTION" CONTENT="Code To Predict Cattle Futures "/>

```

The edit and add methods also compute search keys for each piece of content with the `AddSearchKeys` method of the `SearchUtility` class. The method takes the title and description for a piece of content and breaks each into individual words. The method passes each word to the `Community_SearchAddSearchKey` stored procedure. The stored procedure first checks to see if the word is a noise word (by looking in the `Community_SearchNoiseWords` table), and if not, inserts the word into `Community_SearchKeys`. The noise word check removes words such as "the", "and", and "to" from the searchable word list.

Earlier we examined how the stored procedure to add content would always return the ID of the new content record. We now see how the middle-tier components use this identifier in adding the search keys.

Get Articles

The last two methods in `ArticleUtility` we cover are the methods to retrieve a single article and a list of articles. These methods are similar because they open a connection to the CSK database, attach parameters to a `SqlCommand` object, and then execute a stored procedure. The `GetArticleInfo` method needs to create a single object from the resultset and return the object.


```

SqlDataReader dr = cmdGet.ExecuteReader();
if (dr.Read())
    _articleInfo = new ArticleInfo(dr);
conPortal.Close();
return (ContentInfo) _articleInfo;

```

GetArticles on the other hand needs to return a list of articles. All modules in the CSK use the ArrayList class to store a list of objects. The ArrayList allows easy iteration, and is also a valid and lightweight data source for data-binding controls.

```

ArrayList colArticles = new ArrayList();
conPortal.Open();
SqlDataReader dr = cmdGet.ExecuteReader();
while (dr.Read())
    colArticles.Add(new ArticleInfo(dr));
conPortal.Close();
return colArticles;

```

At this point we have seen all the code required to interact with the underlying data in the CSK tables. The utility classes retrieve records from stored procedures and create info components. Each info component can construct itself given a SqlDataReader object. The next step is to move information from the info components to the presentation level.

Content

Content classes are the code-behind pages for ASPX files in the CSK. Before we can dig into the code, let's take a brief review of what happens when a user navigates to an article. As an example, let's say a user navigates to <http://localhost/CommunityStarterKit/Mission/3.aspx>, which is the URL to an article.

We know this request will first hit the Application_BeginRequest event handler in the CommunitiesModule HttpHandler. The method will inspect the incoming path and determine which community the user is trying to reach by looking at the domain, subdomain, and host pieces of the incoming URL. The CSK can then populate a CommunityInfo object to store in Context.Items collection.

Once the CSK has determined which community the request is for, the next step is to inspect the URL again and determine which section the request is for by looking at the path information—in this case the section is Mission. The handler retrieves information about the Mission section and populates a SectionInfo object to keep in Context.Items. For the purpose of this discussion, the most important property of the section information is the Skin property. For a default CSK installation, the Mission section skin property will have the value Common/Themes/Default.

The last step for the CSK is to inspect the path again and determine what page the user is trying to reach. The CSK parses a 3 from the URL and finds a matching contentPage_id in Community_ContentPages, which gives the CSK enough information to build a PageInfo object to place into the Context.Items collection. The CSK now rewrites the internal request path to the communityDefault.aspx page from where the runtime processes all content requests.

communityDefault.aspx

The default community page takes care of a number of items, including writing out meta tags. The most important job this page performs is loading the page skin and the page content. Let's review the following lines of code from `communityDefault.aspx.cs`:

```
// Load the Page Skin
objPageSkin = LoadControl(
    CommunityGlobals.AppPath +
    "/Communities/" +
    objSectionInfo.Skin +
    "/Skins/PageSkins/Default.ascx" );
// Add the Page Content Page Part
objPagePart = (Placeholder)objPageSkin.FindControl(
    "content" );
if (objPagePart != null)
{
    Control objPageContent =
        (Control)Activator.CreateInstance(
            Type.GetType( objPageInfo.Content, true ) );
    objPagePart.Controls.Add( objPageContent );
}
```

The first step is to dynamically load a control: the page skin control. Given the default installation and the URL we looked at before, the skin file should resolve to `Communities/Common/Themes/Default/Skins/PageSkins/Default.ascx`. This skin gives us the default layout for the page: where the menu appears, where the logo appears, and where the content appears. The skin and the style together customize nearly every visual aspect of the CSK content.

The skin does not know what content will appear. The content could be a list of books or a single article. Every `Default.ascx` page skin needs to define a placeholder where the content will appear similar to the following:

```
<div class="Content">
    <asp:Placeholder id="Content" Runat="Server" />
</div>
```

The `communityDefault.aspx` code will find the content placeholder control and dynamically load a web control into the spot with the `Activator` class. The `Activator` class requires a parameter of type `System.Type`. We can construct the type object from the name of the object we wish to load. The name of the object is kept in the `PageInfo` object's `Content` property, which ultimately came from the `pagetype_pageContent` column in the `Community_PageTypes` table. Given a content page ID, you can find the content page type using the following SQL query:

```
SELECT pagetype_pageContent FROM Community_PageTypes PT
INNER JOIN Community_ContentPages PG ON
    PG.contentPage_pageType = PT.pageTypeID
WHERE PG.contentPage_ID = 3
```

The result for `contentPage_ID=3` is `ASPNET.StarterKit.Communities.Articles.Article`. The code passes this string to the `GetType` method, which returns a `Type` object for the `Activator` class. The `Activator` finds this class in the CSK assembly and invokes the default constructor to create an instance of the class. Finally, the code can add this new object to the `Controls` collection of the placeholder and the control will appear on the page in the correct position. This is the beauty of the CSK—controls and skins are dynamically loaded in data-driven code. The two important pieces, the skin and the content, are outlined in the following figure:



Now we understand how an `Article` object appears on the web page, and the ultimate purpose of the `PageInfo` and `SectionInfo` objects prepared by the `HttpModule`. We will see more of these objects, however, and we still need to detail how the `Article` object works.

Article Content Control

Content controls come in many forms. Examples of content controls in the CSK include a control to allow users to log in, a control to let users register, and, of course, a control to display an article. Most of the module controls such as the one to display an article and the one to display a book are extremely similar.

These controls move information from the `ContentInfo` derived objects to the screen, and they load their own skin. Since these classes are so similar, we again find a class hierarchy where reusable, generic code exists in a base class for other classes to derive from. The following figure shows the inheritance hierarchy the `Article` content class builds on:



All of the content controls, from Login to Article, derive from the `SkinnedCommunityControl` class, which in turn derives from the .NET Framework `WebControl` class. This class takes care of a number of mundane chores; for example, the constructor retrieves references to all of the objects stored in the `HttpContext` Items collection:

```

public SkinnedCommunityControl()
{
    if (Context != null)
    {
        objUserInfo =
            (UserInfo)HttpContext.Current.Items[ "UserInfo" ];
        objPageInfo =
            (PageInfo)HttpContext.Current.Items[ "PageInfo" ];
        objSectionInfo =
            (SectionInfo)HttpContext.Current.Items[ "SectionInfo" ];
        objCommunityInfo =
            (CommunityInfo)HttpContext.Current.Items[
                "CommunityInfo" ];
    }
}
  
```

A `SkinnedCommunityControl` object's primary purpose is to find and load the ASCX file (skin) for itself. This skin file will contain all of the controls required to display the content the user is looking for. The content may be an article or the user registration form. The class is an abstract class, meaning you cannot directly instantiate an object of type `SkinnedCommunityControl`. Instead, you must derive from the class and override any abstract members. We will focus on three properties and one method of the class to see how it locates the correct skin file.

The `SkinName` property represents which skin directory to look in for the ASCX file. The `SkinName` could point to any of the directories underneath `Communities\Common\Themes` (Default, Arc, Cruise, Eco, Frida, Library, Professional, or Robotico). Of course, you can add your own theme directory in customizing the CSK. The `SkinName` property defaults to the skin settings for the current section (obtained from the `sectionInfo` object).

The `SkinType` property of `SkinnedCommunityControl` is an abstract property. Only derived classes will know their skin type, which will let the object know where to locate the ASCX file. If you look in the skins directory for any theme, you'll see four directories, which represent the four skin types: `ContentSkins`, `ControlSkins`, `PageSkins`, and `TemplateSkins`.

Finally, the `SkinFileName` property will contain the name of the ASCX file to load. For example, an `Article` object will set the `SkinFileName` property to `Articles_Article.ascx`.

The `LoadSkin` method and the methods it calls weave all these properties together to load the skin file. The `LoadSkin` method will first try to load the skin from the current theme directory (set by the `SkinName` property).

For example, suppose the Mission section of the CSK used the Arc theme. `LoadSkin` would attempt to find the `Articles_Article.ascx` file underneath the `Communities\Common\Themes\Arc\Skins\ContentSkins` directory. However, requiring a CSK theme designer to provide a skin for each and every control used by the CSK would be a burden, so `LoadSkin` has a failover plan. If the skin file is not found in the theme directory for the current skin name, `LoadSkin` will look in the default skin specifically, `Communities\Common\Themes\Default\Skins\ContentSkins`. In this case, there is no skin for an `Article` underneath the Arc theme, so the CSK will failover to the default directory.

Now, if a designer likes the default layout, he or she can still customize the control colors (and to some extent the layout) with the cascading style sheet in effect for the section, but most importantly the designer doesn't need to replicate every skin file (84 total) underneath the default theme into every other new theme.

However, if there are a handful of controls the designer would like to change, he or she can simply create the ASCX file for the control and place it into the proper directory for the theme; this skin file will override the skin file in the default theme directory. The logic in the following methods is the soul of the CSK and its easy-to-customize feature set:

```
protected Control LoadSkin()
{
    Control skin;
    // Build the skin key
    string skinKey = BuildSkinKey();
    // If skin doesn't exist, failover to Default
    if (Context.Cache[skinKey] != null)
    {
        skinKey = BuildDefaultSkinKey();
        Context.Trace.Warn( "Failover: " + skinKey );
    }
    // Attempt to load the control.
    try
    {
        skin = Page.LoadControl(
            CommunityGlobals.AppPath + skinKey);
    }
    catch (FileNotFoundException fnfEx)
    {
        Context.Trace.Warn( "community error",
            "hard fail file not found: " + skinKey, fnfEx );

        // Add a marker we can check for to skip
        // this in the future
        Context.Cache.Insert(skinKey, "fnf",
            new CacheDependency(
```

```

        Page.MapPath(
            CommunityGlobals.AppPath
            + skinKey));
    // Load Default Skin
    skinKey = BuildDefaultSkinKey();
    try
    {
        skin = Page.LoadControl(
            CommunityGlobals.AppPath + skinKey);
    }
    catch
    {
        throw
            new Exception( "Could not failover to " +
                CommunityGlobals.AppPath + skinKey );
    }
    }
    return skin;
}

private string BuildSkinKey()
{
    // Do we have a skin file?
    if (_skinFileName == null)
        throw new Exception("The SkinFileName " +
            "property of the control must be set.");
    // Do we have a skin name?
    if (_skinName == null)
        _skinName = objSectionInfo.Skin;
    return ("/Communities/" + _skinName +
        "/Skins/" + SkinType + "/" +
        _skinFileName).ToLower();
}

private string BuildDefaultSkinKey()
{
    // Do we have a skin file?
    if (_skinFileName == null)
        throw new Exception("The SkinFileName " +
            "property of the control must be set.");
    return ("/Communities/Common/Themes/Default/Skins/"
        + SkinType + "/" + _skinFileName).ToLower();
}

```

The first step is to build the skin key. The skin key is just a string containing the path and file name to the skin file using the theme directory in effect for the current section. The method looks into the cache collection to see if an object reference exists with this string as the key—we will see exactly why soon. For now, let's assume it does not find a cache entry.

The next step would be to attempt to load the skin file using `Page.LoadControl`. There is a good chance this call may fail—because, as we discussed earlier, themes other than the default theme are not required to carry every skin file. If an exception is thrown because the file does not exist, the method next inserts a small marker object (the string `fnf`) into the cache with the skin key.

Now the early cache check should make more sense. If we were writing the code in English, it would say, "If you find a cache entry with the skin key it means the file doesn't exist, so don't attempt to load the file. Instead, look in the default directory."

This check is here because throwing an exception is a relatively expensive operation. If we were always checking outside the default directory for a skin file, the code would throw an enormous number of exceptions and suffer a performance penalty.

The other interesting feature in `LoadSkin` is the use of a `CacheDependency` object.

A `CacheDependency` object is provided by the .NET Framework and has the ability to remove an entry from the cache collection when it detects a change in the environment.

In this case, `CacheDependency` points to the (non-existent) skin file, but if the file does appear it will remove the marker from the cache. This means you can drop a new skin file into a theme while the application runs; the next time `LoadSkin` executes looking for that particular skin, it will go back and check outside the default directory.

The `ContentItemPage` class derives from `SkinnedCommunityControl` and forms the base of all module content items, for example, `Article`, `Book`, `Event`, `Download`, and so on. The `ContentItemPage` class contains common code for all these classes:

```
public abstract class ContentItemPage :
    SkinnedCommunityControl
{
    public GetContentItemDelegate GetContentItem;
    override protected string SkinType
    {
        get { return "ContentSkins"; }
    }

    override protected void InitializeSkin(Control skin)
    {
    }

    override protected void OnInit(EventArgs e)
    {
        ContentInfo _contentInfo =
            GetContentItem(
                objUserInfo.Username, objPageInfo.ID);
        Context.Items["ContentInfo"] = _contentInfo;
    }
}
```

`ContentItemPage` is still an abstract class, so again you'll need a derived class to use the functionality inside. The class sets the `SkinType` property to return `ContentSkins`, which is the proper directory for any of the module skins. It also provides an implementation of the `InitializeSkin` method, which is an abstract method in the base class. The method body here is empty.

Finally, the class provides a delegate, `GetContentItem`, which will retrieve the `ContentInfo` object during the `OnInit` method. The code will place the `ContentInfo` object into the `Context.Items` collection for the child controls to find.

This delegate must be assigned to in a derived class to call the proper method (the `Article` class will do this). The delegate has the follow declaration:

```
public delegate ContentInfo
    GetContentItemDelegate(string username, int pageID);
```

A delegate points to, or refers to, other methods. When you invoke a delegate, you are really calling the method assigned to the delegate. More than one method can be assigned to a delegate, but the delegate here will only need to refer to one method. This delegate declaration specifies the method signature for any target method, so you can only assign methods to this delegate that have

a return type of `ContentInfo` and accept a string and an int parameter list. The `Article` class below fills in the final details of the control, including assigning a method for the delegate to call:

```
public class Article : ContentItemPage
{
    string _skinFileName = "Articles_Article.ascx";

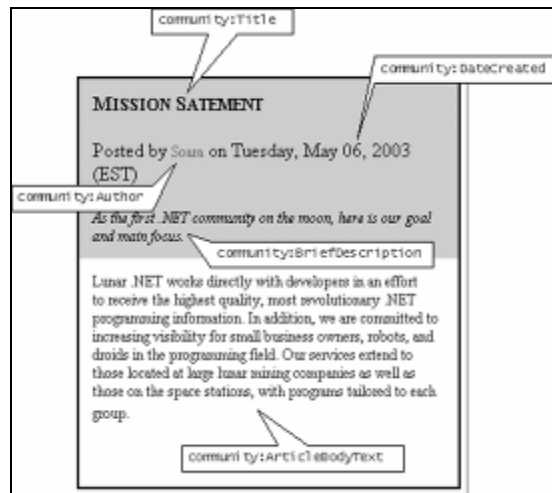
    GetContentItemDelegate _getContentItem =
        new GetContentItemDelegate(
            ArticleUtility.GetArticleInfo);

    public Article() : base()
    {
        SkinFileName = _skinFileName;
        GetContentItem = _getContentItem;
    }
}
```

Since the base classes are capable of performing most of the work, the `Article` class has little to do. The class initializes the `SkinFileName` property and sets the `GetContentItem` delegate to point to the `GetArticleInfo` method, which we reviewed earlier. Before we look at some of the other ways to use a `SkinnedCommunityControl`, let's look at the article skin and how the information displays.

The Article Skin

The article skin lays out web controls to display an article. The following figure shows us the name and position of each control in the default article skin:



Each control knows what text to display by looking into the `Context.Items` collection. In the `Items` collection, the control can either find the `PageInfoObject` or the `ContentInfoObject`. Remember, the `OnInit` method of the `ContentItemPage` object placed a `ContentInfo` object into the container, while the `CommunitiesModule`'s `IHandler` placed the `PageInfo` object in the collection. Here is the code for the `Title` web control:

```

public class Title : WebControl
{
    string _text = String.Empty;
    public Title() : base()
    {
        // assign default css class
        CssClass = "title";
        // Get PageInfo object
        if (Context != null)
        {
            PageInfo objPageInfo =
                (PageInfo)Context.Items["PageInfo"];
            _text = objPageInfo.Title;
        }
    }
    public string Text
    {
        get {return _text; }
        set {_text = value; }
    }
    override protected void RenderContents(
        HtmlTextWriter writer)
    {
        // we need the section for the transformations
        SectionInfo objSectionInfo =
            (SectionInfo)Context.Items["SectionInfo"];
        // display the content
        writer.Write(
            CommunityGlobals.FormatText(
                AllowHtml.None,
                objSectionInfo.ID, _text));
    }
}

```

The Title control pulls information from the PageInfo object and writes the title into the stream of HTML during the RenderContents method. Remember, the title is a property on every PageInfo object because every piece of content has a title.

Let's look at the ArticleBodyText control, which will display the full text of an article. The actual text is article-specific, and is only kept in an ArticleInfo object. The ArticleBodyText control needs to go to the ArticleInfo object stored by ContentItemPage during OnInit:

```

public class ArticleBodyText : WebControl
{
    private string _text;
    public ArticleBodyText() : base()
    {
        CssClass = "articleBodyText";
        // Get ContentInfo object
        if (Context != null)
        {
            Object objArticleInfo =
                Context.Items["ContentInfo"];
            if (objArticleInfo != null)
            {
                _text =
                    ((ArticleInfo)objArticleInfo).BodyText;
            }
        }
    }
    public string Text
    {
        get {return _text; }
        set {_text = value; }
    }
    override protected void
    RenderContents(HtmlTextWriter writer)

```

```
    {
        // we need the section for the transformations
        SectionInfo objSectionInfo =
            (SectionInfo)Context.Items["SectionInfo"];
        // display the content
        writer.Write(
            CommunityGlobals.FormatText(
                objSectionInfo.AllowHtmlInput,
                objSectionInfo.ID, _text));
    }
}
```

As you can see, the code for `ArticleBodyText` follows the same pattern as the `Title` control. In fact, most of these small web controls do the same. The skin file manages the placement of the controls to allow for drastically different layouts. Meanwhile, each control sets the `CssClass` property to allow finer customization through stylesheets.

InitializeSkin Method

Earlier we glossed over the `InitializeSkin` method. The `SkinnedCommunityControl` class marks this method as abstract, and `ContentItemPage` provides a concrete implementation, albeit empty because content pages do not need to take any steps to initialize the skin. In other sections of the CSK, however, this method has an important role.

The reason a content page does not need to initialize the skin is that the web controls on a content page skin carry out their jobs with no assistance. They simply retrieve the information they need from `Context.Items` and then render the required HTML.

There are other skins where the logic behind the page needs to interact with the controls. For instance, to add (and edit) an article uses the same skin, `Articles_AddArticle`. The code behind this skin needs to retrieve values from the controls and pass the values along to the database to save an article. The code also allows the user to preview the content, which requires toggling certain areas of the page between visible and hidden. The only way to interact with the controls is to have a reference to the controls, and this is the real purpose of `InitializeSkin` method.

The `SkinnedCommunityControl` class provides two methods to help wire up a web control variable:

- `GetControl`: Will throw an exception if it does not find the control specified
- `GetOptionalControl`: Will return `NULL` for the same case

Derived classes can use these methods to find controls on the skin. For example, the following code is from `ContentEditPage`:

```
override protected void InitializeSkin(Control skin)
{
    // Find and hide Add Button
    btnAdd = (Button)GetOptionalControl(skin, "btnAdd");
    if (btnAdd != null)
        btnAdd.Visible = false;
    // Find Edit Button
    btnEdit = (Button)GetControl(skin, "btnEdit");
    btnEdit.Click += new EventHandler(OnSubmit);
    // Find Topics Panel
    pnlTopics = (Panel)GetOptionalControl(
        skin, "pnlTopics");
}
```

```

        if (pnlTopics != null)
            if (!objSectionInfo.EnableTopics)
                pnlTopics.Visible = false;

        // Find Preview Panel
        pnlPreview = (Panel)GetOptionalControl(
            skin, "pnlPreview");
        if (pnlPreview != null)
        {
            // Find Form Panel
            pnlForm = (Panel)GetControl(skin, "pnlForm");

            // Find Preview Button
            btnPreview = (Button)GetControl(
                skin, "btnPreview");
            btnPreview.Click += new EventHandler(OnPreview);

            // Find Continue Button
            btnContinue = (Button)GetControl(
                skin, "btnContinue");
            if (btnContinue != null)
            {
                btnContinue.CausesValidation = false;
                btnContinue.Click += new
                    EventHandler(OnContinue);
            }

            // Hide the Preview Panel on first load
            if (!Page.IsPostBack)
                pnlPreview.Visible = false;
        }
        // Call Skin Load event
        OnSkinLoad(this, new SkinLoadEventArgs(skin));
    }

```

ContentEditPage is a base class for EditArticle and the other edit pages for the rest of the modules. Just like ContentItemPage, it contains code to take care of the work that all content editing pages must do, like finding the preview button and assigning an event handler for the click event. Most of the work in the method is finding controls, although the method can and does perform other tasks, such as hiding the preview panel. At the end, the method fires an event to notify any listeners who want to perform additional initialization.

Skins and Styles

Each module needs to provide skins for all content pages in the module. Generally, a module will provide a skin to:

- Edit or add content
- Display a single piece of content
- Display a list of the content

For articles, these skins are named Articles_AddArticle, Articles_Article, and Articles_ArticleSection.

The strategy each skin follows is to break any piece of dynamic content out into a web control. Static text (Posted by, for example) you can find placed directly into the ASCX, while a web control will always display an author name.

The following is an excerpt from `Articles_Article`:

```
<%@ Control %>
<%@ Register TagPrefix="Community"
    Namespace="ASPNET.StarterKit.Communities"
    Assembly="ASPNET.StarterKit.Communities" %>

<table width="100%" cellpadding="0" cellspacing="11"
    class="Article_Table">
  <tr>
    <td class="Article_IntroCell">
      <div align="right">
        <community:DisplayTopic
          Runat="Server" />
      </div>
      <community:Title
        CssClass="Article_Title"
        Runat="Server" />
      <br>
      <br>
      Posted by
      <community:Author
        CssClass="Article_AuthorLink"
        Runat="Server" />
      on
      <community:DateCreated Runat="Server" />
      <br>
      <br>
      <community:BriefDescription
        CssClass="Article_BriefDescription"
        Runat="Server" />
    </td>
  </tr>
```

There will always be, at a minimum, a full set of skins in the `Default` theme directory. The CSK also provides some customized skins for some of the other themes.

Each control in a skin generally gets its own CSS class. For example, in the preceding code there exist an `Article_Title` CSS class and an `Article_AuthorLink` CSS class. This allows the title display in the author module to look different from the title display in the book module.

Other important areas in each module may also get their own CSS class. For example, the table cell with the article introduction has a distinct CSS class (`Article_IntroCell`) allowing one to differentiate the introduction from the full text of the article. The article module requires a total of 19 styles be defined in the stylesheet for a theme.

Summary

In this chapter we have reviewed the `Articles` module and seen the overall blueprint for module building in the CSK. By using generic tables, common stored procedures, and an object-oriented hierarchy of classes, the CSK is able to produce more functionality with less code.

The architecture lends itself to building additional functionality, as we shall see in Chapter 8. The architecture also lends itself to easy customization through the use of skins and styles. The heart of the CSK can be found in the `CommunitiesModule` (which determines what content the user is trying to reach and sets instructions accordingly) and the `SkinnedCommunityControl` (which reads the instructions to carry out the orders and load content).

8

Creating a New Module

The next three chapters in this book will demonstrate various methods of customizing the CSK. Every community site will have different requirements to fulfill. Although the existing CSK framework offers a great deal of flexibility, having the entire source code available means you can add additional functionality to a site in an elegant manner. In this chapter, we will concentrate on creating a new module for the CSK. We will see how creating a new module allows you to add entirely new features which integrate seamlessly with the rest of the framework. In this chapter, we will implement a **Frequently Asked Questions (FAQ)** module.

Before we begin, let's mention one caveat. The CSK is a living piece of software. It will undoubtedly gain additional features and modules from the developer community, so one question you may want to answer is, "Has someone else already written the module I need?" Once you've made the commitment to customizing the CSK with your own code, you'll need to also think about integrating your code into newer versions of the CSK. If you stick to the current design used by the existing modules, chances are you'll find that the upgrades are easier.

Module Design

Before you begin implementing a new module for the CSK, you will first want to have a firm grasp of the features you wish to add, and then decide if any of the existing modules shipped with the CSK can fulfill that functionality.

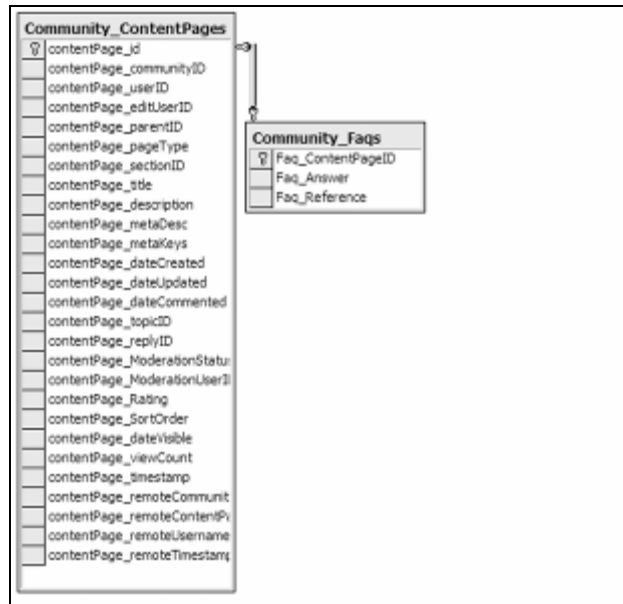
First, let us make a brief list of requirements for our community FAQ:

- An FAQ should consist of a question, an answer, a description or introduction, and pointers to additional references.
- Community users should have the ability to comment on and rank individual FAQs, as well as offer e-mail notifications when a new FAQ appears.
- Community users should have the ability to submit a new FAQ subject for the moderator's approval.

You could certainly create a list of questions and answers marked up in HTML and add the content to a community site using the HTML Page section type. However, the HTML Page section type offers limited user interaction (no comments, ranking, e-mails, or moderation).

Alternatively, the Articles section type could provide us with what we need, if we are willing to lump the FAQ answer and reference fields together in the article's body text. For maximum flexibility in presenting information, we would prefer to keep these as distinct entities. With our requirements and direction set, let's take a look at the classes and tables we will be building.

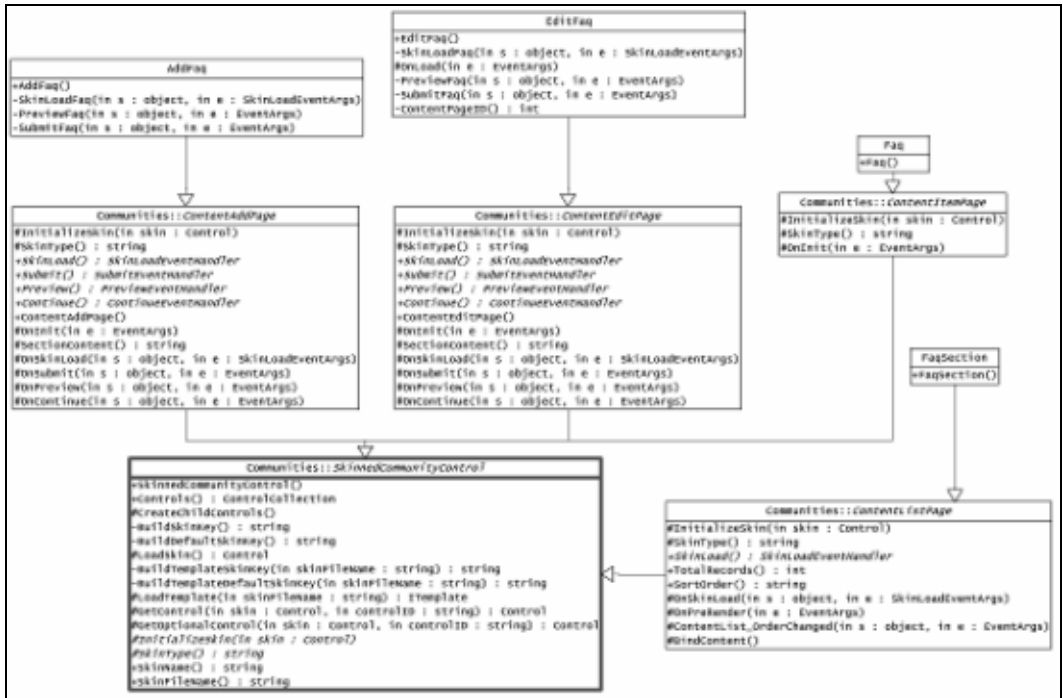
We know from the earlier chapters that the `Community_ContentPages` table will keep most of the information we need for an FAQ; for example, the author name, view count, and description. If we consider the question piece of the FAQ as the title, we really only need to store the FAQ answer and additional references as attributes. We will add a database table (`Community_Faqs`) as shown in the following diagram:



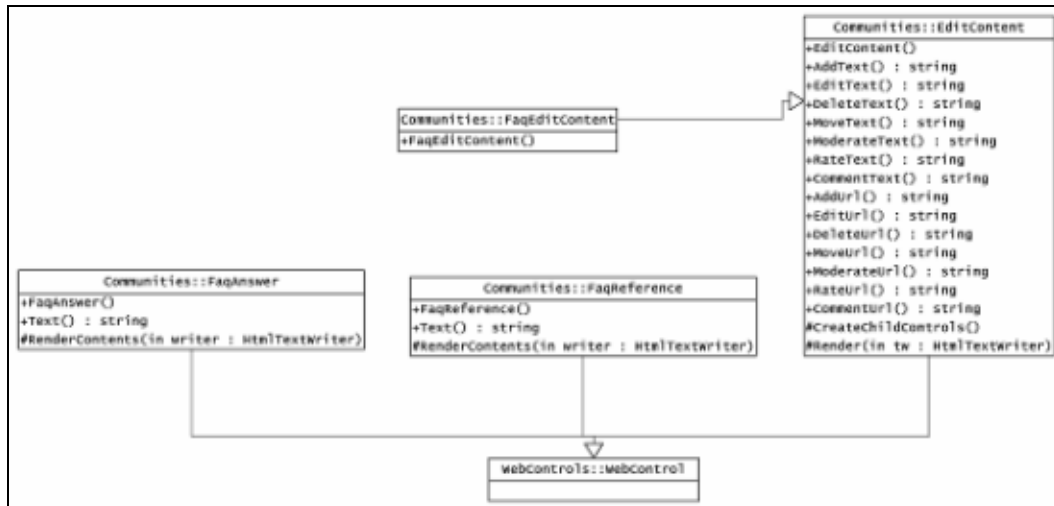
We can then build a class to hold FAQ information. As shown in the following diagram, the `FaqInfo` class inherits from `ContentInfo`, which holds most of the attributes for any content item. Every module also uses a utility class to retrieve, add, and edit content. For the FAQs module, this is the `FaqUtility` class (shown without method parameters).



We will also need to build classes for the code-behind pages that display and edit FAQs. We saw in earlier chapters how pages in the CSK derive from the `SkinnedCommunityControl` to allow themselves to be displayed with different skins. There are also a number of base classes with most of the behavior we need to add, edit, and display FAQs. The following diagram shows the class hierarchy that we will use for the FAQ code-behind classes:



We will also need to create `webControl`-derived classes to display FAQ content. Typically, each attribute of our content will display in a distinct control, which allows a skin to lay out the content in whatever manner it sees fit. The following diagram shows the controls that we will use for this module, all of which ultimately derive from `webControl`:



Module Construction Blueprint

We will take a bottom-up approach by starting with the database work, and finishing with presentation skins and themes. We will follow the naming conventions and patterns established by the existing modules in the CSK so that the code fits well with the rest of the framework. For example, the Books module retrieves book information from the `Community_Books` table to populate a `BookInfo` component.

Thus we will use a table called `Community_Faqs` to populate an `FaqInfo` component. However, you may want to consider adding a unique identifier in case a future version of the CSK contains the module you are building. For instance, if you work for ABC Inc. you might use `Community_ABCFaqs` as a table name to lower the possibility of future name collisions.

We will use the following steps to build the FAQ module. You can also follow these steps in a general sense to build your own modules:

16. Create a new table (`Community_Faqs`) to hold the additional fields for the new module.
17. Create stored procedures to add, edit, and select a single FAQ, and a stored procedure to select all FAQs for a given section.
18. Create a maintenance stored procedure to initialize the FAQ module by populating the `Community_PageTypes` and `Community_NamedPages` tables.
19. Create a component (`FaqInfo`) to hold information on a single FAQ.

20. Create a component (Faquility) with data-access routines to invoke our FAQ-related stored procedures.
21. Create controls derived from `WebControl` to display the individual module fields. These controls will be `FaqQuestion`, `FaqIntro`, `FaqAnswer`, `FaqReference`, and `FaqEditContent`.
22. Create controls derived from the `SkinnedCommunityControl` class to contain the logic behind the page content skins from the next step. These controls will be `AddFaq`, `EditFaq`, `FaqSection`, and `Faq`.
23. Create new page-content skins for the module. This includes `Faqs_AddFaq.ascx`, `Faqs_FaqSection.ascx`, and `Faq_Faq.ascx`. We will use the `Faqs_AddFaq` skin to add a new FAQ as well as edit an existing FAQ. At a minimum you will need to create a default skin in the `ContentSkins` folder under the `Communities\Common\Themes\Default\Skins` directory. You can optionally create additional skins for other available themes, such as the `Robotico` and `Professional` themes.
24. Create style rules in the CSS files in `Communities\Common\Themes\Default\Styles` for the module. You should also add CSS rules into all of the CSS files in the themes your community may use.

The rest of this chapter will explain each step in more detail.

The Community_Faq Table

Content that is common to all modules such as title, description, and view count resides in the `Community_ContentPages` table. Additional module-specific content requires a second table for storage. For the FAQ module, we can store the FAQ question in the `contentPage_title` field of `Community_ContentPages`, and the FAQ introduction in the `contentPage_description` field. We still need to store the FAQ answer and the additional references text for the FAQ, so we will use the following DDL to create a table:

```
CREATE TABLE [Community_Faqs] (
    [Faq_ContentPageID] [int] NOT NULL ,
    [Faq_Answer] [ntext] NOT NULL ,
    [Faq_Reference] [ntext] NULL,
    CONSTRAINT [PK_Community_Faqs] PRIMARY KEY CLUSTERED
    (
        [Faq_ContentPageID]
    ),
    CONSTRAINT [FK_Community_Faqs_Community_ContentPages]
    FOREIGN KEY
    (
        [Faq_ContentPageID]
    ) REFERENCES [Community_ContentPages] (
        [contentPage_id]
    ) ON DELETE CASCADE
)
```

The naming conventions and data types we use for the table are consistent with the other modules shipped with the CSK.

We store the answer and reference data in fields of type ntext in order to support large quantities of text (up to 1 GB). Also notice how `Faq_Answer` is a required field but `Faq_Reference` may contain a NULL value. Our primary key (`Faq_ContentPageID`) references the additional FAQ content in the `Community_ContentPages` table. The foreign key constraint will ensure our referential integrity for this relationship.

Another nice design feature is the use of cascading referential integrity restraints. The `ON DELETE CASCADE` clause in the foreign key constraint means we will not need to write any code to delete an FAQ record from `Community_Faqs`.

The CSK uses the `Community_ContentPagesDeleteContentPage` stored procedure to delete records from the `Community_ContentPages` table. When the procedure removes a record from the content table, SQL Server will automatically remove the corresponding entry from the FAQ table.

The Community_Faqs Stored Procedures

The next steps in our blueprint calls for us to create stored procedures to add an FAQ, edit an FAQ, retrieve a single FAQ, and retrieve a paged and sorted list of FAQs, for a total of four procedures. All of the data access in the CSK happens via stored procedures. There is no ad hoc SQL in the code, which is a good practice from both encapsulation and security standpoints. The first procedure we will write is to add new FAQ content.

Community_FaqsAddFaq

The stored procedure for adding a new FAQ to the database is shown now. We do not need to provide a parameter for every column in the two content tables. For example, we do not need to pass a parameter to populate the `contentPage_ViewCount` column. Many of the columns contain a sensible default value (`contentPage_ViewCount` defaults to 0) or allow NULL values (`contentPage_dateCommented`) for new content.

```
CREATE PROCEDURE Community_FaqsAddFaq
(
    @communityID int,
    @sectionID int,
    @username nvarchar(50),
    @topicID int,
    @question nvarchar(100),
    @introduction nvarchar(500),
    @metaDescription nvarchar(250),
    @metaKeys nvarchar(250),
    @moderationStatus int,
    @answer ntext,
    @reference ntext
)
AS
DECLARE @ContentPageID int
DECLARE @pageType int
SET @pageType = dbo.Community_GetPageTypeFromName('Faq')
DECLARE @userID int
SET @userID = dbo.Community_GetUserID(@communityID, @username);
BEGIN TRAN
EXEC @ContentPageID = Community_AddContentPage
    @communityID,
    @sectionID,
    @userID,
```

```

        @question,
        @introduction,
        @metaDescription,
        @metaKeys,
        @pageType,
        @moderationStatus,
        @topicID

INSERT Community_Faqs
(
    Faq_ContentPageID,
    Faq_Answer,
    Faq_Reference
)
VALUES
(
    @ContentPageID,
    @answer,
    @reference
)
COMMIT TRAN
RETURN @ContentPageID

```

Notice the use of two UDFs supplied with the CSK. The first UDF retrieves the page type for the content. There is a distinct page type identifier for each module (Books, Articles, Downloads, and so on) in the CSK. We will discuss page types in more detail when we create our maintenance stored procedure. A second UDF allows us to retrieve the `userID` by passing a `communityID` and `username`.

Since we must insert the content items into two different tables we use a transaction to make the operation atomic. Inserting records into the `Community_ContentPages` table occurs by invoking the `Community_AddContentPage` procedure passing the FAQ question as the `@Title` parameter and the FAQ introduction as the `@Description` parameter. `Community_AddContentPage` returns the primary key value of the newly inserted record which we will in turn use in the `INSERT` command for `Community_Faqs`.

All of the procedures that add new content records must return the primary key value of the new record as a result.

The new `contentPageID` value will be useful in the upper layers of software, as we will see when we write the data-access component.

Community_FaqsEditFaq

The stored procedure we use to edit an existing FAQ uses a slightly different parameter list. Obviously, some columns are immutable after we add a content page to the system (such as the section identifier). The edit procedure listing is shown below:

```

CREATE PROCEDURE Community_FaqsEditFaq
(
    @communityID int,
    @contentPageID int,
    @username NVarchar(50),
    @topicID int,
    @question NVarchar(100),
    @introduction NVarchar(500),

```

```
        @metaDescription NVarchar(250),
        @metaKeys NVarchar(250),
        @answer Text,
        @reference Text
    )
AS
DECLARE @UserID int
SET @UserID = dbo.Community_GetUserID(@communityID, @username)
EXEC Community_EditContentPage
    @contentPageID,
    @userID,
    @question,
    @introduction,
    @metaDescription,
    @metaKeys,
    @topicID
UPDATE Community_Faqs SET
    Faq_Answer = @answer,
    Faq_Reference = @reference
WHERE Faq_ContentPageID = @contentPageID
```

Again, we use a stored procedure provided by the CSK to update the `Community_ContentPages` pages, and then follow with an `UPDATE` statement for the `Community_Faqs` table. Unlike the procedure to add a new FAQ record, there is no transaction present to keep the two table updates atomic. We are following a pattern established in the existing module edit procedures in the CSK—none of these use a transaction. Presumably the designers found the consequences of a failure during a content edit to be considerably smaller compared to the same during content addition. We should have slightly better system throughput by avoiding SQL resource locks.

Community_FaqsGetFaqs

The next stored procedure to write is the procedure to return all FAQs for a given section in a community. The `Community_GetPagedSortedContent` UDF, which we covered earlier in the book, should essentially dictate the parameter list required to retrieve content. All we need to do is augment the resultset of the UDF with some FAQ-specific columns and sort by the `IndexID` column the UDF computes.

```
CREATE PROCEDURE Community_FaqsGetFaqs
(
    @communityID int,
    @username NVarchar(50),
    @sectionID int,
    @pageSize int,
    @pageIndex int,
    @sortOrder NVarchar(50)
)
AS
DECLARE @currentDate DATETIME
SET @currentDate = GetUtcDate()
SELECT
    null Faq_Answer,
    null Faq_Reference,
    Content.*
FROM
    dbo.Community_GetPagedSortedContent
    (
        @communityID,
        @username,
        @sectionID,
        @currentDate,
```

```

        @sortOrder,
        @pageSize,
        @pageIndex,
        default
    ) Content
ORDER BY
    IndexID

```

The stored procedure uses a couple of techniques to reduce the amount of code we need to write and to reduce the amount of maintenance required in future changes. First, we use `Content.*` in the `SELECT` list to return all columns in the UDF resultset. This code is consistent with the other stored procedures within the CSK. In terms of efficiency, it is better to explicitly list all the columns required instead of having the database engine figure out the available columns. However, in this case, the CSK designers decided to lean towards maintainability. With this code you can make certain types of schema changes to the `Community_ContentPages` (adding a column, for example) and not need to modify and test any of the stored procedures (almost 30) working with records from this table.

The second thing to point out is the addition of two empty columns (`Faq_Answer` and `Faq_Reference`) to the resultset. Later, we will write the `FaqInfo` component to hold results of both this stored procedure *and* the next stored procedure that retrieves a single FAQ. Since we want to use the same component for both operations, we need to populate the resultset with all the columns an `FaqInfo` object expects to see. As these two columns can be quite large, and would never display in a summary list of FAQs, we do not want to use up resources moving these columns around for the FAQ list, we just set the values to `NULL`.

Community_FaqsGetFaq

The stored procedure we write to retrieve the content for a single FAQ also has one other responsibility. It needs to increment the view count for the content page and indicate the user has now read this content page. Both these tasks are accomplished by executing the CSK's `Community_ContentPagesTrackStats` procedure. The entire procedure is shown here:

```

CREATE PROCEDURE Community_FaqsGetFaq
(
    @communityID INT,
    @username NVarchar(50),
    @contentPageID int
)
AS
DECLARE @userID INT
SET @userID = dbo.Community_GetUserID(@communityID, @username)
-- Update ViewCount and HasRead Stats
EXEC Community_ContentPagesTrackStats @userID, @contentPageID
DECLARE @currentDate DATETIME
SET @currentDate = GetUtcDate()
SELECT
    Faq_Answer,
    Faq_Reference,
    Content.*
FROM
    dbo.Community_GetContentItem(
        @communityID,
        @userID,
        @currentDate) Content

```

```

        JOIN Community_Faqs (nolock)
            ON ContentPage_ID = Faq_ContentPageID
WHERE
    ContentPage_ID = @contentPageID

```

Note that this time we actually retrieve the value for the `Faq_Answer` and `Faq_Reference` columns, as they should display at this level of detail. This requires a join to our `Community_Faqs` table. You'll see that we include a locking hint of `nolock`, which allows dirty reads but avoids any contention for the content we retrieve.

Initializing the FAQ Module

Every community module has a corresponding maintenance stored procedure to populate the database with settings required for the module to work. Specifically, we need to register content page types by inserting two records into `Community_PageTypes`: one for an FAQ section page (to display a list of FAQs) and one for an FAQ page (showing a single FAQ in detail). We call the procedure `Community_MaintenanceInitializeFaqs`, following the existing CSK naming convention. An excerpt registering the FAQ section page type is shown here:

```

IF NOT EXISTS (SELECT * FROM Community_PageTypes WHERE pageType_Name='Faq
Section')
BEGIN
    INSERT Community_PageTypes
    (
        pageType_name,
        pageType_description,
        pageType_pageContent,
        pageType_IsSectionType,
        pageType_ServiceSelect
    )
    VALUES
    (
        'FAQ Section',
        'Contains FAQs in a question and answer style format',
        'ASPNET.StarterKit.Communities.Faqs.FaqSection',
        1,
        'Community_FaqsServiceSelect'
    )
END
ELSE
    PRINT 'WARNING: The FAQ Module has already been registered.'

```

The CSK caches data from `Community_NamePages` so as to retrieve the data only once. If you make modifications to the table, you'll need to restart the web application for the changes to take effect in the CSK.

The maintenance stored procedure also needs to register the named pages (static content) for the new module. Named pages for the FAQ section will include the page to add an FAQ and a page to edit an FAQ. You'll have to choose your page names at this point and use the same names later when you create the ASPX file.

Here's an excerpt from `Community_MaintenanceInitializeFaqs` to add a named page for adding FAQs:

```
IF NOT EXISTS (SELECT * FROM Community_NamedPages WHERE
namedPage_Path='/Faqs_AddFaq.aspx')
BEGIN
    INSERT Community_NamedPages
    (
        namedPage_name,
        namedPage_path,
        namedPage_pageContent,
        namedPage_title,
        namedPage_description,
        namedPage_sortOrder,
        namedPage_isVisible,
        namedPage_menuID
    )
VALUES
(
    'AddFaq',
    '/Faqs_AddFaq.aspx',
    'ASPNET.StarterKit.Communities.Faqs.AddFaq',
    'Add FAQ',
    'Enables users to add a new FAQ',
    0,
    1,
    0
)
END
ELSE
    PRINT 'WARNING: /Faqs_AddFaq.aspx has already been registered as a
NamedPage.'
```

The `namedPage_pageContent` parameter is the name of the class that the CSK will instantiate as the code-behind logic for the page. The name includes the full namespace qualifier `ASPNET.StarterKit.Communities.Faqs.AddFaq`.

The maintenance stored procedure needs to execute during the database setup. We will take a look at how to do this in Chapter 11.

FAQ Components

The C# code for our FAQ module will reside in the `Engine\Modules\Faqs` directory. First, we will write out helper components and place these in a `Components` directory. Each module in the CSK places components inside a distinct namespace below `ASPNET.StarterKit.Communities`, and the existing modules use the name of the module as the additional namespace qualifier (Faqs).

FaqInfo

`FaqInfo` class extends the `ContentInfo` class to offer data properties specific to an FAQ. The code for this class is shown as follows:

```
using System;
using System.Data.SqlClient;
namespace ASPNET.StarterKit.Communities.Faqs
{
```

```
public class FaqInfo : ContentInfo
{
    public FaqInfo(SqlDataReader dr) : base(dr)
    {
        if(dr["Faq_Answer"] != DBNull.Value)
        {
            _answerText = (string)dr["Faq_Answer"];
        }
        if(dr["Faq_Reference"] != DBNull.Value)
        {
            _referenceText = (string)dr["Faq_Reference"];
        }
    }
    public string AnswerText
    {
        get { return _answerText; }
        set { _answerText = value; }
    }
    public string ReferenceText
    {
        get { return _referenceText; }
        set { _referenceText = value; }
    }
    public string QuestionText
    {
        get { return base.Title; }
        set { base.Title = value; }
    }
    public string IntroText
    {
        get { return base.BriefDescription; }
        set { base.BriefDescription = value; }
    }
    private string _answerText;
    private string _referenceText;
}
}
```

FaqInfo expects initialization with an instance of the SqlDataReader class. We will be writing the data-access code to create a SqlDataReader in our next class.

FaqUtility

Following the patterns set forth in the rest of the CSK, we will put all of our data-access routines into static methods of a utility class. There should be one static method available for each of the FAQ-related stored procedures (with the exception of the maintenance stored procedure, which we should not need to invoke during regular operations of the community site but only during setup). Each of these routines will need to map incoming variables to stored procedure parameters and execute the procedure.

Here's the AddFaq method:

```
public static int AddFaq(
    string username,
    int sectionID,
    int topicID,
    string question,
    string introduction,
    string answer,
    string reference,
    int moderationStatus)
{
}
```

```

SqlConnection conPortal = new SqlConnection(
    CommunityGlobals.ConnectionString);
SqlCommand cmdAdd = new SqlCommand(
    "Community_FaqsAddFaq", conPortal);
cmdAdd.CommandType = CommandType.StoredProcedure;
cmdAdd.Parameters.Add("@RETURN_VALUE",
    SqlDbType.Int).Direction =
    ParameterDirection.ReturnValue;
cmdAdd.Parameters.Add("@communityID",
    CommunityGlobals.CommunityID);
cmdAdd.Parameters.Add("@sectionID", sectionID);
cmdAdd.Parameters.Add("@username", username);
cmdAdd.Parameters.Add("@topicID", topicID);
cmdAdd.Parameters.Add("@question", question);
cmdAdd.Parameters.Add("@introduction", introduction);
cmdAdd.Parameters.Add("@metaDescription",
    ContentPageUtility.CalculateMetaDescription(introduction));
cmdAdd.Parameters.Add("@metaKeys",
    ContentPageUtility.CalculateMetaKeys(introduction));
cmdAdd.Parameters.Add("@moderationStatus", moderationStatus );
cmdAdd.Parameters.Add("@answer", SqlDbType.NText);
cmdAdd.Parameters.Add("@reference", SqlDbType.NText);
cmdAdd.Parameters["@answer"].Value = answer;
cmdAdd.Parameters["@reference"].Value = reference;

conPortal.Open();
cmdAdd.ExecuteNonQuery();
int result = (int)cmdAdd.Parameters["@RETURN_VALUE"].Value;
SearchUtility.AddSearchKeys(conPortal, sectionID, result,
    question, introduction);

conPortal.Close();
return result;
}

```

Notice that the AddFaq method also generates the search keys for the content using the SearchUtility class, and the newly created identifier of the content returned by the stored procedure we reviewed earlier.

The EditFaq method almost duplicates the AddFaq method except for calling a different stored procedure and using EditSearchKeys on the SearchUtility class to update the FAQ search keys.

One improvement you might consider making to the CSK is adding a try catch finally statement to ensure the database connection will always invoke the Close method, even in the face of an exception. The chances of an exception are small, but on a high volume community site, you cannot afford the opportunity to waste database connections.

The other two methods in FaqUtility are GetFaqs and GetFaqInfo. GetFaqs loops through records in a SqlDataReader to return an ArrayList of FaqInfo objects, while GetFaqInfo expects only a single record in the database results and returns a single new FaqInfo object. These two methods from the class are shown here:

```

public static ContentInfo GetFaqInfo(string username, int contentPageID)
{
    FaqInfo faq = null;
    SqlConnection conPortal = new SqlConnection(
        CommunityGlobals.ConnectionString);
    SqlCommand cmdGet = new SqlCommand(
        "Community_FaqsGetFaq", conPortal);
    cmdGet.CommandType = CommandType.StoredProcedure;
    cmdGet.Parameters.Add(
        "@communityID", CommunityGlobals.CommunityID);
    cmdGet.Parameters.Add("@username", username);
    cmdGet.Parameters.Add("@contentPageID", contentPageID);
}

```

```
        conPortal.Open();
        SqlDataReader dr = cmdGet.ExecuteReader();
        if (dr.Read())
            faq = new FaqInfo(dr);
        conPortal.Close();
        return faq;
    }
    public static ArrayList GetFaqInfo(string username, int sectionID,
                                      int pageSize, int pageIndex, string sortOrder)
    {
        SqlConnection conPortal = new
        SqlConnection(CommunityGlobals.ConnectionString);
        SqlCommand cmdGet = new SqlCommand( "Community_FaqsGetFaqInfo",
        conPortal);
        cmdGet.CommandType = CommandType.StoredProcedure;
        cmdGet.Parameters.Add("@communityID",
            CommunityGlobals.CommunityID);
        cmdGet.Parameters.Add("@username", username);
        cmdGet.Parameters.Add("@sectionID", sectionID);
        cmdGet.Parameters.Add("@pageSize", pageSize);
        cmdGet.Parameters.Add("@pageIndex", pageIndex);
        cmdGet.Parameters.Add("@sortOrder", sortOrder);

        ArrayList faqs = new ArrayList();
        conPortal.Open();
        SqlDataReader dr = cmdGet.ExecuteReader();
        while (dr.Read())
            faqs.Add(new FaqInfo(dr));
        conPortal.Close();
        return faqs;
    }
}
```

It is important for `GetFaqInfo` to use the return value and parameter list shown above. The framework should invoke these methods through a delegate and the signatures must match. We will see how this works when we write the content pages.

Our data-access layer is now complete. If you build a module in this fashion, you should be able to compile the solution at this time to resolve any errors. You might consider writing a driver page to exercise the four static methods in `FaqUtility` and verify the results by looking in the `Community_Faqs` and `Community_ContentPages` tables of the database.

FAQ WebControls

The CSK breaks up the display of content into smaller controls. For example, under `Engine\Framework\ContentPages\Controls`, you'll find a control to display the content title (in `Title.cs`), and the content's brief description (`BriefDescription.cs`), which can display our FAQ question and the introduction. All we will need to add are a couple of controls specific to the FAQ module: a control to display the answer and the reference, and a control to provide a link for authorized users to edit the FAQ content.

FaqAnswer and FaqReference

All of the controls at this level derive from the .NET Framework `WebControl` class. We simply need to set the `CssClass` property for our control, retrieve the text to display from the current `HttpContext`, and override the `RenderContents` method to write the text.

Create these controls in the `Engine\Module\Faqs\Controls` directory. The control to display the answer to an FAQ is as follows:

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using ASPNET.StarterKit.Communities.Faqs;
using System.ComponentModel;
namespace ASPNET.StarterKit.Communities
{
    [Designer(typeof(ASPNET.StarterKit.Communities.CommunityDesigner))]
    public class FaqAnswer : WebControl
    {
        public FaqAnswer() : base()
        {
            CssClass = "faqAnswerText";
            if(Context != null)
            {
                Object faqInfo = Context.Items["ContentInfo"];
                if(faqInfo != null)
                {
                    _text = ((FaqInfo)faqInfo).AnswerText;
                }
            }
        }
        public string Text
        {
            get { return _text; }
            set { _text = value; }
        }
        override protected void RenderContents(
            HtmlTextWriter writer)
        {
            SectionInfo objSectionInfo =
                (SectionInfo)Context.Items["SectionInfo"];
            writer.Write(
                CommunityGlobals.FormatText(
                    objSectionInfo.AllowHtmlInput,
                    objSectionInfo.ID, _text));
        }
        private string _text;
    }
}
```

First, notice that our previous components were in the `ASPNET.StarterKit.Communities.Faqs` namespace, but we place our control in a higher level namespace. This remains consistent with the rest of the CSK where the skin files typically set up a `Community` tag prefix pointing to the `ASPNET.StarterKit.Communities` namespace. We can place the above component into a skin with a line similar to:

```
<community:FaqAnswer Runat="Server" ID="Answer1"
    NAME="Answer1"/>
```

The `FaqReference` control looks very similar to the `FaqAnswer` control. Both override `RenderContents` and use the `CommunityGlobals` class for proper transformation and formatting of the outgoing text. Generally, you will want to break each field of a new module into a specific control to use from a content skin. It is up to the skin designer to decide where to lay out the fields for presentation.

FaqEditContent

Every CSK module uses a control derived from `EditContent` to display links for authorized users to add, delete, move, comment, and moderate content. All we need to do is set the appropriate URL properties. The logic in the base class will determine when to display the appropriate links. We perform all of the work in the constructor, as follows:

```
public FaqEditContent()
{
    if (Context != null)
    {
        PageInfo pageInfo = (PageInfo)Context.Items["PageInfo"];
        int contentPageID = pageInfo.ID;
        AddUrl = "Faqs_AddFaq.aspx";
        EditUrl = String.Format(
            "Faqs_EditFaq.aspx?id={0}",
            contentPageID);
        DeleteUrl = String.Format(
            "ContentPages_DeleteContentPage.aspx?id={0}",
            contentPageID);
        MoveUrl = String.Format(
            "ContentPages_MoveContentPage.aspx?id={0}",
            contentPageID);
        CommentUrl = String.Format(
            "Comments_AddComment.aspx?id={0}",
            contentPageID);
        ModerateUrl = "Moderation_ModerateSection.aspx";
    }
}
```

With these small pieces built we are ready to tackle the actual content display.

Content Classes

In the traditional ASP.NET paradigm, the content classes are the *code-behind* files. Since the CSK takes a slightly different tack to allow high levels of customization, we will not be able to use the IDE to keep our web form in sync with the code behind the form. No real tie exists between the two, since each code file supports multiple versions of the same web form (the *skins*). Instead, we will need to manually keep track of which controls are on the page, and manually wire up the events that we need. The task is not difficult but does require some extra attention to control names and other details.

We have four content classes to write for the four pages we will need for our FAQ module:

- `Faq`: To display a single FAQ item
- `FaqSection`: To display a list of FAQs
- `AddFaq`: For the data entry of FAQ content
- `EditFaq`: For the update of FAQ content

The amount of code you will need to write for a content class varies widely. Using the `ContentItemPage` and `ContentListPage` CSK classes we can display an FAQ and an FAQ list with very little code. We will look at these two classes first.

Faq and FaqSection

The `FaqSection` class derives from `ContentListPage`, which can do most of our work with just a little more of information.

```
public class FaqSection : ContentListPage
{
    public FaqSection() : base()
    {
        SkinFileName = _skinFileName;
        GetContentItems = _getContentItems;
    }
    string _skinFileName = "Faqs_FaqSection.ascx";
    GetContentItemsDelegate _getContentItems =
        new GetContentItemsDelegate(FaqUtility.GetFaqs);
}
```

At this point, we need to pick the actual filename for our skin and set the `SkinFileName` property. This step is essential to `SkinnedCommunityControl` (the base class of `ContentListPage`) for finding the correct skin to load.

When we wrote the data-access methods in `FaqUtility`, we mentioned the need to maintain a specific method signature of return type and parameters. The method signature we used was the one for a `GetContentItemsDelegate` delegate. The base class will use the delegate within the `BindContent` method to retrieve and display all the FAQs in a given section. The `Faq` class follows the same pattern, but initializes the skin file to `Faqs_Faq.ascx`, and assigns the delegate to the `FaqUtility.GetFaq` method.

AddFaq and EditFaq

These two classes present a bit of a challenge. Since the data-access methods to insert and update module content vary widely from module to module, there is no base class available to reduce the workload through a delegate. Instead, we will need to find controls that are specific to our module to get and set values, and invoke the `FaqUtility` data-access routines in response to user events.

Before reaching this point, you may want to sketch out a skin file to know what controls you will need on the page. We know the skin file that we will use to add and edit FAQs will need the following input controls:

- `TextBox`: For the FAQ question
- `TextBox`: For the FAQ introduction
- `TopicPicker`: For FAQ sections supporting topics
- `HtmlTextBox`: For the FAQ answer
- `HtmlTextBox`: For the FAQ references

In addition, we would like to preview the control, which requires five more controls for display instead of input. These five controls should be the same as the ones we will use in the display of an FAQ. So we will use the `FaqAnswer` control we wrote earlier to display the answer. Let us look at the `EditFaq` class as an example.

The constructor is as follows:

```
public EditFaq() : base()
{
    SkinFileName = _skinFileName;
    SectionContent = _sectionContent;

    this.SkinLoad += new SkinLoadEventHandler(SkinLoadFaq);
    this.Preview += new PreviewEventHandler(PreviewFaq);
    this.Submit += new SubmitEventHandler(SubmitFaq);
}
```

The constructor initializes the skin file name and section content properties, which we will define later. The constructor then wires up event handlers for three events defined in the base class `ContentEditPage`. These event handlers will contain the logic for loading the skin, handling the preview button click, and the submit button click. They are a part of every content edit page.

```
void SkinLoadFaq(Object s, SkinLoadEventArgs e)
{
    txtQuestion = (TextBox)GetControl(e.Skin, "txtQuestion");

    // continue initializing all controls with GetControl . . .
}
```

As we discussed in earlier chapters, the CSK dynamically loads a skin (ASCX) file that lays out the controls for a particular theme. If you need to programmatically interact with any of the controls on a skin, you'll need to obtain a reference to the control. When editing an FAQ we will need to obtain the contents of the `TextBox` object holding the FAQ question. You can obtain references to controls using the `GetControl` and `GetOptionalControl` methods implemented in the `SkinnedCommunityControl` base class. There are additional controls we will need reference to, but only the question `TextBox` is shown here:

```
protected override void OnLoad(EventArgs e)
{
    if (!Page.IsPostBack)
    {
        ContentPageID = Int32.Parse(
            Context.Request.QueryString["id"]);

        FaqInfo faqInfo =
            (FaqInfo)FaqUtility.GetFaqInfo(
                objUserInfo.Username, ContentPageID);

        EnsureChildControls();
        txtAnswer.Text = faqInfo.AnswerText;
        dropTopics.SelectedTopicID = faqInfo.TopicID;
        txtIntro.Text = faqInfo.IntroText;
        txtQuestion.Text = faqInfo.QuestionText;
        txtReference.Text = faqInfo.ReferenceText;
    }
}
```

When the page loads, we need to retrieve the information for an existing FAQ from the database. The CSK will pass the content identifier in the query string parameters, so we fetch the ID and pass it along to the `GetFaqInfo` method of the `FaqUtility` class we examined earlier. Once we have an `FaqInfo` object in hand, we can populate controls on the page (which we obtained references to during the skin load event) with FAQ information.

```

void PreviewFaq(Object s, EventArgs e)
{
    if(objSectionInfo.EnableTopics)
        topicPreview.Name = dropTopics.SelectedItem.Text;
    questionPreview.Text = txtQuestion.Text;
    introductionPreview.Text = txtIntro.Text;
    answerPreview.Text = txtAnswer.Text;
    referencePreview.Text = txtReference.Text;
}

```

When the user clicks the preview button, we need to shuttle all of the content in the edit controls to the preview controls, which will render the content with the styles we use to display content. This gives the author a better idea of how the content will look when the content goes live. The ContentEdit base class will take care of toggling the preview panel control visibility so the author can see the results.

```

void SubmitFaq(Object s, EventArgs e)
{
    if (Page.IsValid)
    {
        // Get Topic
        int topicID = -1;
        if (objSectionInfo.EnableTopics)
            topicID = Int32.Parse(dropTopics.SelectedItem.Value);

        FaqUtility.EditFaq(
            objUserInfo.Username,
            objSectionInfo.ID,
            ContentPageID,
            topicID,
            txtQuestion.Text,
            txtIntro.Text,
            txtAnswer.Text,
            txtReference.Text);

        Context.Response.Redirect(CommunityGlobals.CalculatePath(
            String.Format("{0}.aspx", ContentPageID)));
    }
}

```

The SubmitFaq event handler uses the FaqUtility class to put the updated content into the database. Once this is done, we send the user off to the content page with a Response.Redirect to view the updated FAQ.

```

int ContentPageID
{
    get { return (int)ViewState["ContentPageID"]; }
    set { ViewState["ContentPageID"] = value; }
}
TextBox txtQuestion;
TopicPicker dropTopics;
TextBox txtIntro;
HtmlTextBox txtAnswer;
HtmlTextBox txtReference;
DisplayTopic topicPreview;
Title questionPreview;
BriefDescription introductionPreview;
FaqAnswer answerPreview;
FaqReference referencePreview;
string _skinFileName = "Faqs_AddFaq.ascx";
string _sectionContent =
    "ASPNET.StarterKit.Communities.Faqs.FaqSection";
}

```

The AddFaq page looks similar to the EditFaq page and can be seen in the code download. The only work left is in presenting the data via skins and style sheets.

FAQ Page Content Skins

Our FAQ module requires three skins:

- A skin to display a single FAQ in detail
- A skin to display a list of FAQs for a section
- A skin to add or edit an FAQ

At a minimum, we need to create these three skin files in the default theme for the communities. We can add additional variations of the skins to the other themes if we want the FAQs to take on a different appearance. For adding and editing an FAQ, we would like the screen to look like the following:

The screenshot shows a web form titled "LUNAR FAQ" with the instruction "Use this form to add or edit a FAQ." The form is divided into three main sections: "QUESTION:", "ENTER TEXT:", and "ANSWER:". The "QUESTION:" section has a single-line text input. The "ENTER TEXT:" section has a multi-line text input with the prompt "Enter a brief description of the contents of the FAQ". The "ANSWER:" section has a multi-line text input with the prompt "Enter the answer for the FAQ". Above the "ANSWER:" input is a rich text editor toolbar with buttons for bold, italic, underline, bulleted list, numbered list, link, unlink, image, and other formatting options. Below the toolbar are dropdown menus for "Normal" (font style) and "Times New Roman" (font face), and a "3" (font size) dropdown. The entire form is enclosed in a light gray border.

Make sure the filename for the skin matches the filename assigned to `SkinFileName` in the content page class. You'll also need to pay special attention to control names, as these must match the control names you search for with `GetControl` in the underlying class.

The easiest way to get started is with an existing skin from a working module, because you'll also need to match up the controls to the base class. Remember that our `AddFaq` class derives from the `ContentAddPage`, which expects certain controls on the form, such as a panel named `pnlForm`, and a button named `btnAdd`. Let's look at an excerpt from the `Faq_AddFaq` skin:

```
<%@ Control %>
<%@ Register TagPrefix="community"
```

```

        Namespace="ASPNET.StarterKit.Communities"
        Assembly="ASPNET.StarterKit.Communities" %>
<community:SectionTitle
    CssClass="Form_Title"
    Runat="Server"
    ID="Sectiontitle1"
    NAME="Sectiontitle1"/>
<p class="Form_Description">
    Use this form to add or edit an FAQ.
</p>
<asp:Panel id="pnlForm" Runat="Server">
<TABLE cellSpacing="0" cellPadding="3" width="520" class="Form_Table">
    <TR>
        <TD class="Form_SectionRow">
            FAQ Form
        </TD>
    </TR>
    <tr class="Form_LabelRow">
        <td >
            <span class="Form_LabelText">Question:</span>
            <asp:RequiredFieldValidator
                ControlToValidate="txtQuestion"
                Text="(Required)"
                Runat="Server"
                ID="Requiredfieldvalidator1"
                NAME="Requiredfieldvalidator1"/><br>
            <asp:TextBox id="txtQuestion"
                CssClass="Form_Field"
                columns="40"
                runat="server">
            </asp:TextBox>
        </td>
    </tr>
    ...

```

Note the use of a `RequiredFieldValidator` to ensure the FAQ will have a question populated. The `ContentAddPage` class will also expect a preview panel, which comes later in the `Faq_AddFaq.ascx` file.

```

...
<asp:Panel id="pnlPreview" Runat="Server">
<table cellpadding="5" width="520">
<tr>
    <td align="right">
        <community:DisplayTopic id="topicPreview" runat="Server"/>
    </td>
</tr>
<tr>
    <td>
        <community:Title id="questionPreview" Runat="Server"/>
    </td>
</tr>
<tr>
    <td>
        <community:BriefDescription id="introductionPreview"
            Runat="Server" />
    </td>
</tr>
<tr>
    <td>
        <community:FaqAnswer id="answerPreview" Runat="Server" />
    </td>
</tr>

```

```

<tr>
  <td>
    <community:FaqReference id="referencePreview" Runat="Server" />
  </td>
</tr>
</table>
<p>
  <asp:Button id="btnContinue" Text="Finish Preview" Runat="Server" />
</asp:Panel>
...

```

The skins to display content are easier to build since you just need to lay out the display controls as you see fit. The base content display pages will match up the controls for you. Here is the skin to display a single FAQ:

```

<%@ Control %>
<%@ Register
  TagPrefix="community"
  Namespace="ASPNET.StarterKit.Communities"
  Assembly="ASPNET.StarterKit.Communities" %>
<table width="100%" cellpadding="0" cellspacing="11" class="Faq_Table">
  <tr>
    <td class="Faq_IntroCell">
      <div align="right">
        <community:DisplayTopic Runat="Server"
          ID="Displaytopic1" NAME="Displaytopic1" />
      </div>
      <community:Title Runat="Server" ID="Title1"
        NAME="Title1" />
      <br>
      <br>
      Posted by
      <community:Author CssClass="Faq_AuthorLink"
        Runat="Server" ID="Author1" NAME="Author1" />
      on
      <community:DateCreated Runat="Server"
        ID="Datecreated1" NAME="Datecreated1" />
      <br>
      <br>
      <community:BriefDescription Runat="Server"
        ID="Introduction1" NAME="Introduction1" />
    </td>
  </tr>
  <tr>
    <td class="Faq_AnswerCell">
      <community:FaqAnswer Runat="Server"
        ID="Answer1" NAME="Answer1" />
    </td>
  </tr>
  <tr>
    <td class="Faq_AnswerCell">
      <community:FaqReference Runat="server"
        ID="Reference1" Name="Reference1" />
    </td>
  </tr>
  <tr>
    <td class="Faq_BodyCell">
      <br>
      <community:Rating SubmitText="Rate Item"
        Runat="Server" ID="Rating1" NAME="Rating1" />
    </td>
  </tr>
</table>

```

```

<table width="100%" cellpadding="11">
  <tr>
    <td>
      <div class="Content">
        <community:Notify
          Text="Notify me when a new comment is posted"
          Runat="Server" ID="Notify1" NAME="Notify1" />
        <p>
          <community:Comments Runat="Server"
            ID="Comments1" NAME="Comments1" />
        <p>
          <community:FaqEditContent
            CommentText="Add Your Comment"
            EditText="Edit this FAQ"
            DeleteText="Delete this FAQ"
            Runat="Server" ID="Faqeditcontent1"
            NAME="Faqeditcontent1" />
        </div>
      </td>
    </tr>
  </table>

```

In this skin, we use all of the web controls we built (including `FaqEditContent` which only displays links appropriate to the type of user viewing the FAQ). With the skins in place, we are only one step away from completing our new module.

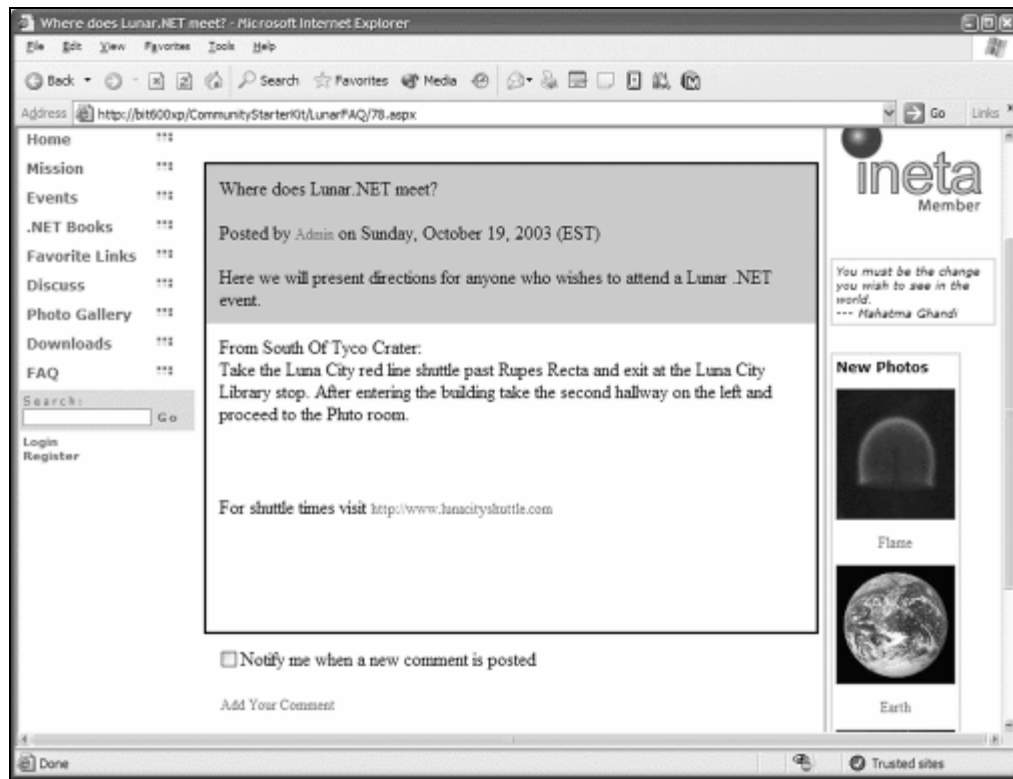
Module Styles

While building the skins and web controls for a module, keep track of the different CSS rule names you place into the code. The time has finally come to modify CSS files to put the new styles required by the module into place. Unfortunately, the CSK doesn't provide any default behavior for stylesheet selection, so you will need to place these styles into every stylesheet available to a community.

Copy styles from an existing module and just change the names.

Putting It Together

At this time, you can start testing the new module. The working result of the FAQ module is displayed in the following image:



Summary

Building a new module requires building on top of existing code in the CSK, so a good understanding of the CSK architecture is in order. By following the steps we outlined in the beginning of this chapter and following the patterns set forth by the existing modules, you can build a new module with the ability to search and rate the module content. It will also have all of the other cross-cutting functionality built into the CSK.

In the next chapter, we will build a little more onto the FAQ module and take a look at other forms of customization in the CSK. Although building a new module requires a bit of code and time, the end result is a seamlessly integrated feature with the full support of the CSK comments, ratings, view counts, and more.

9

Customization with User Controls

A lightweight technique for customizing the CSK with new functionality and content is through user control files. The CSK Framework allows you to add custom modules and Web Boxes by simply placing a user control file (.ascx) in the appropriate directory for a community.

In contrast to the last chapter—where we added components, utility classes, and skin files—user controls are relatively self-contained. All of the logic (except perhaps a stored procedure) and markup resides in the ASCX file itself. Since ASP.NET compiles ASCX files on the fly, you do not need to rebuild the solution to have them up and running.

In this chapter, we will take a look at implementing a custom module section and various Web Boxes for a community.

The Custom Module Section

Before implementing a custom module as a user control, let's discuss some of disadvantages when compared to the approach taken in the last chapter. A custom-module user control does not integrate as closely with the CSK Framework as a fully blown custom module, which ties into searching, sorting, ranking, commenting, and other built-in features. Turn to a user control custom module when you need a small and specific piece of functionality for a community.

As an example, let's give the administrators of a community the ability to view the list of users who have most recently logged in to the system.

The "Recent Logins" User Control

Our custom module will consist of one ASCX file and one stored procedure. The ASCX file for a custom module must reside in the CustomModules directory of a community (for the default CSK Lunar .NET community, this is the Communities\Community1\CustomModules directory). Custom modules are community specific and there is no default directory to fall back on if the CSK does not find the user control file—if you want the module in more than one community, you will need to replicate the file into *all* communities that will use the module.

In keeping with the pattern laid out in the rest of the CSK, all the code that we need to write for our module will stay in the ASCX file (we will not use a code-behind file). Although you *can* use a code-behind file, you will need to recompile the CSK and redeploy the CSK assembly. We will discuss some tips and tricks to make working without code-behind easier. In keeping with the patterns of the CSK, the data will be retrieved from the database using a stored procedure examined next.

Community_RecentLogins Stored Procedure

The CSK stores login information in the `Community_Users` table of the CSK database. The `User_LastLogin` table will contain the exact information we are looking for. Given a community ID, all we need to do is return the last 20 users to have logged in to the community. The code for the stored procedure is shown next:

```
CREATE PROCEDURE Community_RecentLogins
    @communityID int
AS
SELECT TOP 20
    User_UserName,
    User_LastLogin
FROM
    Community_Users WITH(nolock)
WHERE
    User_CommunityID = @communityID
ORDER BY
    User_LastLogin DESC
RETURN
```

If need be, adding additional output fields for display later will be easy. As we will see when we build the user control, making modifications is straightforward when compared to the module we built in the last chapter.

Recent Logins.ascx

A custom user control needs to have exactly the same name as the section name in a community. So, we have called ours `Recent Logins` user control. A custom module must reside in the `CustomModule` directory for a community. If you wanted to add a custom module to a community with the name `Community1`, the ASCX file would be added to the `CustomModules` directory under `CommunityStarterKit\Communities\Community1`. This file can be created with Visual Studio .NET by right-clicking on the `CustomModule` directory in the Solution Explorer window and selecting `Add | Add Web User Control`. However, you'll need to make some changes to the auto-generated code.

First, let's look at the ASP.NET directives we will create in our new user control. Note that these are not the same directives as the ones that the IDE provides:

```
<%@ Control Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="ASPNET.StarterKit.Communities" %>
```

If you create the ASCX file with Visual Studio .NET, make sure you remove the `AutoEventWireup="false"` attribute from the `Control` directive. When you are not using a

code-behind file, `AutoEventWireup` needs to be set to `true` (the default if not specified) in order to catch the `Load` and `Init` events of the control. The ASP.NET runtime will look for specific method names (`Page_Load` and `Page_Init`) in the server-side script to handle these two events.

Another step you'll want to take if you are using the IDE is to click on `Show All Files` in the toolbar of the Solution Explorer window. This will allow you to expand the `Recent Logins.ascx` node and delete the code-behind `.cs` file.

Be careful if you use the Design view of Visual Studio .NET—the designer will sometimes rewrite the `Control` directive and disable `AutoEventWireup` if you switch between Design and HTML views. When working with the CSK, you might find it easier to view all pages with HTML view instead of Design view. You can change the default view under `Tools | Options | HTML Designer | General` in the IDE.

Next, we will look at the server-side code in the ASCX file:

```
<script runat="server">
void Page_Load()
{
    if(!IsPostBack)
    {
        SqlConnection conPortal = new SqlConnection(
            CommunityGlobals.ConnectionString);

        SqlCommand cmdGet = new SqlCommand(
            "Community_RecentLogins", conPortal);
        cmdGet.CommandType = CommandType.StoredProcedure;

        cmdGet.Parameters.Add(
            "@communityID",
            CommunityGlobals.CommunityID);

        conPortal.Open();

        SqlDataReader dr = cmdGet.ExecuteReader();
        LoginList.DataSource = dr;
        LoginList.DataBind();
        conPortal.Close();
    }
}
</script>
```

Our code is fairly standard for the CSK. We open a database connection, invoke our stored procedure, and retrieve the results in a `SqlDataReader` object. Instead of using the CSK's `ContentList` class to display the results; we will just rely on an ASP.NET `DataList` control.

One challenge you might face when writing a user control with no code-behind is the lack of **Intellisense** in the HTML view for C# code. Because ASP.NET compiles the code in an ASCX file at runtime, you will not be able to check for syntax errors until you actually try to view the page in the browser. We will return to this subject later in the chapter; for now, let's take a look at the HTML and ASP.NET control tags in the ASCX file:

```
<p align="center">
    <span class="Search_ListTitle">Recent Logins</span>
</p>
<p align="center">
    <asp:DataList ID="LoginList"
        CssClass="Search_ListTable"
        runat="server">
```

```

<HeaderStyle CssClass="Search_ListTitle">
</HeaderStyle>
<ItemStyle CssClass="Search_ListItem">
</ItemStyle>
<AlternatingItemStyle
    CssClass="Search_ListAlternatingItem">
</AlternatingItemStyle>
<HeaderTemplate>
    <td>Username</td>
    <td>Last Login</td>
</HeaderTemplate>
<ItemTemplate>
    <td>
        <%#
            DataBinder.Eval(Container.DataItem,
                "User_UserName", "{0:s}")
        %>
    </td>
    <td align="right">
        <%#
            DataBinder.Eval(Container.DataItem,
                "User_LastLogin", "{0:d}")
        %>
    </td>
</ItemTemplate>
</asp:DataList>

```

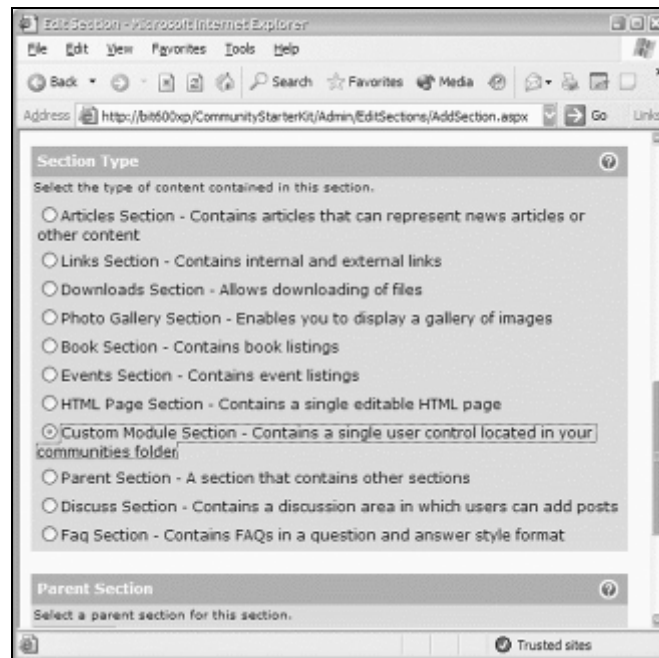
While building the user control, we are making use of CSS rules defined for the search module of the CSK (for example, `Search_ListTitle` and `Search_ListTable`). Since a custom module is specific to the community where the ASCX file resides, it is not important to use only the styles predefined by the CSK. We are using existing styles for convenience and to ensure our module looks somewhat like the rest of the content in our community.

The ASP.NET `DataList` server control will take care of iterating through the data source. All we need to do is provide an `ItemTemplate` to display each record. Inside the template, we use the static `Eval` method of the `DataBinder` class to evaluate our binding expression and format the results. With our stored procedure and ASCX file in place, we can look at how to use the new custom module.

Recent Logins Administration

To add the new custom module, log in as the community administrator and navigate to the Admin section. Click on the Add New Section link in the Edit Sections page. The section name for the custom module must exactly match the name of the ASCX file without the `.ascx` extension. For our example, the section name is Recent Logins.

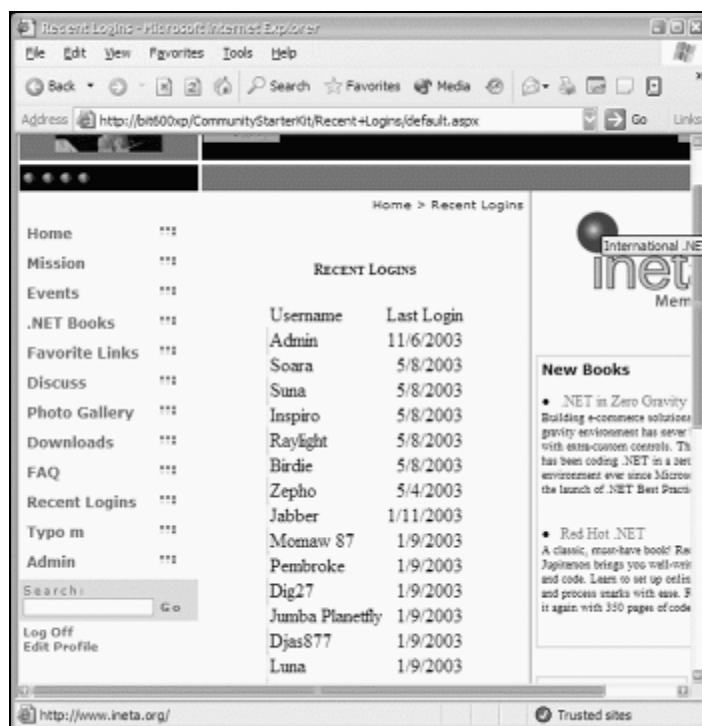
Fill out the rest of the fields as appropriate and make sure to select Custom Module Section as the section type, as shown in the following image:



Keep in mind that we want to restrict this module to community administrators only. Make sure you click on the Security link on the add section page to remove Community-Everyone from the View Roles while adding Community-Administrators. The result should look as follows:



If you navigate back to the home page of the community and click on the new Recent Logins link on the menu, you should see the display similar to the following screenshot:



If the file name of the ASCX file does not match the section name added in the administration screens, you'll see a file not found error at this point. Also, if ASP.NET encounters compilation errors for the C# code in the ASCX file, an error message screen will be displayed.

Now you can begin to experiment some more with the custom module controls, perhaps adding additional fields or changing the format. There is no need to recompile the project between changes—we simply save the ASCX file.

In the next section, we will build some additional user controls to work with the FAQ module from the previous chapter.

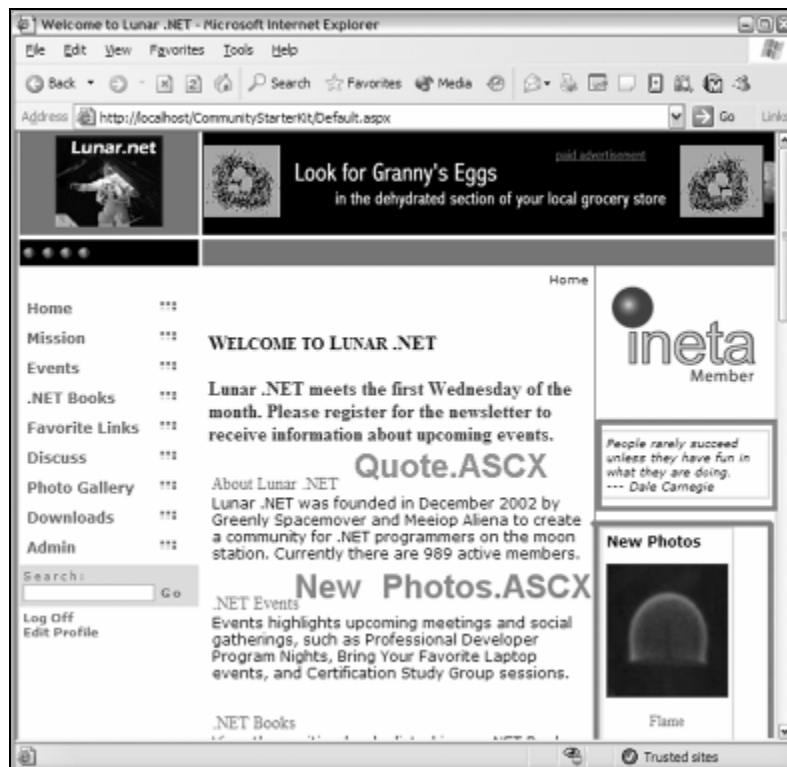
Custom Web Boxes

There are two types of Web Boxes in the CSK:

- The global Web Box, which is available to every community on a site and located in the `Communities\Common\WebBoxes` directory

- The community-specific Web Box, which will reside in the WebBoxes directory underneath a specific community

There are two global Web Boxes we will implement for our FAQ module from the last chapter. These are the "new" and "popular" Web Boxes available for almost every other module (Articles, Books, and Download, to name a few) in the CSK. These Web Boxes are small bits of content that appear in the periphery of the main content window and are a good way to alert users to new content and draw attention to specific areas of a site. In the following screenshot, we have highlighted two Web Boxes, Quote.ASCX (community-specific) and New Photos.ASCX (global):



Let's complete our FAQ module by adding the new FAQ and popular FAQ Web Boxes.

FAQ Web Boxes

Some of the work we need to do to display new and popular FAQs has already been included by the CSK. There are two stored procedures (`Community_GetPopularContent` and `Community_GetNewContent`) to search and return content information given a community ID and a page type name as parameters. The page type name we need to pass to the stored procedure is the name we registered in the `Community_PageTypes` table, specifically, `Faq`.

In addition, wrapper methods to invoke these two stored procedures already exist in the `ContentPageUtility` class. After creating a `New Faqs.ascx` file in the `Communities\Common\WebBoxes` directory, we can add the following code:

```
<%@ Control Language="C#" %>
<%@ Register
    TagPrefix="community"
    Namespace="ASPNET.StarterKit.Communities"
    Assembly="ASPNET.StarterKit.Communities" %>
<%@ Outputcache
    Duration="600"
    VaryByParam="none"
    VaryByControl="ContentList"%>
<%@ Import Namespace="ASPNET.StarterKit.Communities" %>
<script runat="server">

    void Page_Load()
    {
        ContentList.DataSource =
            ContentPageUtility.GetNewContent("Faq");
        ContentList.DataBind();
    }

</script>
<span class="WebBox_TitleText">New FAQs</span>
<br>
<community:ContentList id="ContentList" Runat="Server">
    <ItemTemplate>
        <li>
            <community:ItemTitleLink
                Runat="Server"
                ID="Itemtitlelink1" />
            <br>
            <community:ItemBriefDescription
                Runat="Server"
                ID="Itembriefdescription1" />
            <br>
            <br>
        </li>
    </ItemTemplate>
    <NoContentTemplate>
        <i>There are no new FAQs</i>
    </NoContentTemplate>
</community:ContentList>
```

If you compare the source for `New Faqs.ascx` with any of the existing Web Boxes in the CSK, you will notice a difference in the output cache directive. We have removed the `VaryByParam` attribute and added a `VaryByControl` attribute. This fixes a slight flaw in the CSK. You may notice on occasion how the CSK displays multiple copies of the same Web Box on a page. For example, even if you may have selected to display the New Article and New Books Web Boxes, the CSK may display two New Books boxes instead.

The dynamic loading of Web Boxes coupled with the `VaryByParam="none"` attribute causes ASP.NET to output all of the Web Boxes from the same cache entry—which would be the last Web Box rendered. Using `VaryByControl` and passing the ID of a server-side control on the page will fix this issue. ASP.NET will instead cache output based on the value of the control specified. You may need to make this change to other Web Boxes in the CSK to avoid the problem.

The Web Box to display the most popular FAQs looks extremely similar. The `GetPopularContent` method will invoke a stored procedure and return the results in an `ArrayList` of `ContentInfo` objects.

All of the content will display in the CSK's ContentList control using the ItemTitleLink and ItemBriefDescription controls supplied by the CSK. We just need to set the CSS styles to match those of the other Web Boxes for our community.

```
<%@ Control Language="C#" %>
<%@ Register
    TagPrefix="community"
    Namespace="ASPNET.StarterKit.Communities"
    Assembly="ASPNET.StarterKit.Communities" %>
<%@ Outputcache
    Duration="600"
    VaryByControl="Title"%>
<%@ import Namespace="ASPNET.StarterKit.Communities" %>
<script runat="server">

    void Page_Load()
    {
        ContentList.DataSource =
            ContentPageUtility.GetPopularContent("Faq");
        ContentList.DataBind();
    }

</script>
<span
    class="WebBox_TitleText"
    id="title"
    runat="server">
    Most Popular FAQs
</span>
<br>
<community:ContentList id="ContentList" Runat="Server">
    <ItemTemplate>
        <li>
            <community:ItemTitleLink
                Runat="Server"
                ID="Itemtitlelink1" />

            <br>
            <community:ItemBriefDescription
                Runat="Server"
                ID="Itembriefdescription1" />

            <br>
            <br>
        </ItemTemplate>
        <NoContentTemplate>
            <i>There are no articles</i>
        </NoContentTemplate>
    </community:ContentList>
```

To add the new Web Boxes into the site's display, navigate to the Edit Sections page of the Admin area again. For testing, click on the default page and then on Web Boxes to add the boxes to the first page you see. Remove the existing Web Boxes; you should see the two controls we just created listed as available Web Boxes, as shown in the following screenshot:



Remember, the CSK builds this list by reading *what* ASCX files are present in the Communities\Common\WebBoxes directory. Although this helps you to avoid spelling mistakes when adding Web Boxes to a site, there are still a couple of caveats:

- You may want to remove the output cache directive from a Web Box until you have tested the new control and are happy with the display output. If ASP.NET has the control in cache, the changes you make to the ASCX file will not be reflected until the cache entry expires (as long as 600 seconds for our control) or the application restarts. You can force an application restart by running `iisreset` from the command line, and also by *touching* the `web.config` file (if the timestamp of `web.config` changes, the ASP.NET runtime will restart the website, simply resaving the file will force an update).
- Watch out for compile errors in the ASCX file. Unlike the custom module control we built earlier in the chapter, a Web Box control fails silently if the CSK cannot load the control. To make debugging easier, you may want to temporarily make the failure display an error message in the web browser.

To achieve the error message display mentioned in the second bullet, look at the `WebBoxList.cs` file in the `Engine\Framework\WebBoxes\Controls` directory. This class in this file contains a `Boxes_ItemDataBound` method where the CSK attempts to dynamically load the ASCX file. Change the try catch block around the call to `Page.LoadControl` to resemble the snippet that follows. Now when an error occurs, the code re-throws the exception that will remain unhandled and appear as an error message in the browser.

```

try
{
    // Load the control
    Control box = Page.LoadControl(
        WebBoxUtility.GetWebBoxPath(boxName));

    // Add control to Placeholder
    placeholder.Controls.Add(box);
}
catch (Exception ex)
{
    ActivityUtility.RecordError(
        "Could not find Web Box", ex);
    throw ex;
}

```

When the new Web Boxes are working correctly, we should have a display similar to the following screenshot:



Current Event Web Box

Our final example of a custom Web Box will be a current events calendar. The calendar will display the days of the current month and next month with a clickable link on any date where a community event exists. This Web Box will be community specific, and will thus reside in the `Communities\Community1\WebBoxes` directory.

The first step is to write a stored procedure to return a resultset of current events. The CSK already provides a component to hold the resultset of event-related information, the `EventInfo` class, which is very similar to the `FaqInfo` class we built in the last chapter.

We will need our stored procedure to return all of the columns required to populate an `EventInfo` instance. Much of this work can be performed by the `Community_GetContentItem` UDF, which (unlike what the name suggests) will return multiple content items that you can filter. Let's take a look at the first half of the stored procedure:

```
CREATE PROCEDURE Community_EventsGetCurrentEvents
    @communityID int,
    @username NVarchar(50)
AS
    DECLARE @currentDate DateTime
    SET @currentDate = GetUtcDate()

    DECLARE @currentMonth int
    SET @currentMonth = DATEPART(month, @currentDate)

    DECLARE @UserID Int
    SET @UserID = dbo.Community_GetUserID(
        @communityID, @username)

    DECLARE @pageType INT
    SET @pageType = dbo.Community_GetPageTypeFromName('Event')

    DECLARE @PageIndex TABLE
    (
        PageID int
    )

    INSERT INTO @PageIndex(PageID)
    SELECT
        ContentPage_ID
    FROM
        Community_ContentPages
    WHERE
        ContentPage_communityID = @communityID
        AND ContentPage_PageType = @pageType
        AND ContentPage_ModerationStatus = 1
        AND ContentPage_DateVisible < @currentDate
        AND DATEPART(month, contentPage_SortOrder)
            BETWEEN @currentMonth AND @currentMonth + 1
```

After initializing all the variables that we will use in the stored procedure, we build a table variable of content page IDs. The `WHERE` clause carefully selects only IDs for content in the specified community, with the correct page type (Event), and approved, visible, and in the upcoming date range.

The reason we build the table variable is because of the need to filter on `ContentPage_PageType`, which is not an output column from the `Community_GetContentItem` UDF we call next:

```
SELECT
    null Event_Link,
    null Event_FullDescription,
    Event_Location,
    Event_Speaker,
    null Event_SpeakerBiography,
    null Image_ID,
    null Image_FileName,
    Content.*
```

```

FROM

    dbo.Community_GetContentItem(
        @communityID, @userID, @currentDate) Content

    INNER JOIN Community_Events WITH(nolock)
        ON ContentPage_ID = Event_ContentPageID

    INNER JOIN @PageIndex PI
        ON ContentPage_ID = PI.PageID

ORDER BY
    contentPage_SortOrder DESC

RETURN

```

Here we are filtering the `Community_GetContentItem` records with an `INNER JOIN` to the table variable of content IDs. We also join to the `Community_Events` table to pick up all the columns required by the `EventInfo` class. However, since we will not be using fields such as `Event_FullDescription`, we return `NULL` values to avoid some overhead of unused fields.

In order to invoke this stored procedure, we will add a method to the `EventUtility` class. You can find this class in `EventUtiltiy.cs` under `Engine\Modules\Events\Components`.

```

public static ArrayList GetCurrentEvents()
{
    ArrayList events = new ArrayList();

    SqlConnection connection = new SqlConnection(
        CommunityGlobals.ConnectionString);

    SqlCommand command = new SqlCommand(
        "Community_EventsGetCurrentEvents", connection);
    command.CommandType = CommandType.StoredProcedure;

    command.Parameters.Add("@communityID",
        CommunityGlobals.CommunityID);
    command.Parameters.Add("@username", "none");

    connection.Open();
    SqlDataReader dr = command.ExecuteReader();
    while(dr.Read())
    {
        events.Add(new EventInfo(dr));
    }
    connection.Close();
    return events;
}

```

With an `ArrayList` full of `EventInfo` objects, we can now turn our attention to the `ASCX` file and the controls we will use to display the information. The following code shows the first half of our `ASCX` file:

```

<%@ Control Language="C#" %>
<%@ Register
    TagPrefix="community"
    Namespace="ASPNET.StarterKit.Communities"
    Assembly="ASPNET.StarterKit.Communities" %>
<%@ Outputcache duration="600" VaryByControl="Title"%>
<%@ import Namespace="ASPNET.StarterKit.Communities" %>
<%@ import
    Namespace="ASPNET.StarterKit.Communities.Events" %>

```

```
<span class="WebBox_TitleText"
      id="Title" runat="server">
    Current Events
</span>
<br>
<asp:Calendar
    ID="EventsCalendar1"
    Runat="server"
    ShowNextPrevMonth="False"
    TitleFormat="Month"
    ShowGridLines="True"
    OnDayRender="EventsCalendar_DayRender"
    CssClass="WebBox" >
    <DayHeaderStyle CssClass="WebBox_TitleRow">
    </DayHeaderStyle>
    <OtherMonthDayStyle CssClass="WebBox_SubText">
    </OtherMonthDayStyle>
</asp:Calendar>
<asp:Calendar
    ID="EventsCalendar2"
    Runat="server"
    ShowNextPrevMonth="False"
    TitleFormat="Month"
    ShowGridLines="True"
    OnDayRender="EventsCalendar_DayRender"
    CssClass="WebBox" >
    <DayHeaderStyle CssClass="WebBox_TitleRow">
    </DayHeaderStyle>
    <OtherMonthDayStyle CssClass="WebBox_SubText">
    </OtherMonthDayStyle>
</asp:Calendar>
```

Our Web Box will display two calendar controls, one for the current month and one for the next month. As expected, most of the work here is for setting up the display of the control. Since we are using these calendars to *display* information and not *select* a specific date, we turn off the next and previous month navigation elements in both calendars. Another important piece involves assigning an event handler for the `OnDayRender` events of both the calendars. This will allow us to modify the content of each date cell in the calendar table.

Next, let's turn our attention to the contents of the script tag, where we place our C# code. First, the `Page_Load` event handler:

```
ArrayList events = null;
EventComparer eventComparer = new EventComparer();

void Page_Load()
{
    EventsCalendar1.VisibleDate = DateTime.Now;
    EventsCalendar2.VisibleDate = DateTime.Now.AddMonths(1);
    events = EventUtility.GetCurrentEvents();
    events.Sort(eventComparer);
}
```

The `Page_Load` method ensures that the calendar controls are set to display the correct months, and then retrieves the current events and sorts them with the help of an `EventComparer`. The `Sort` method of the `ArrayList` requires all the elements in the container to implement the `IComparable` interface. If they do not, you must provide an object implementing `IComparer` for the `ArrayList` to use. The `IComparer` interface has only one method, `Compare`, which allows the `ArrayList` to understand if an `EventInfo` is greater than, less than, or equal to a second `EventInfo` object.

Our implementation of the EventComparer in the ASCX file is as follows:

```
public class EventComparer : IComparer
{
    int IComparer.Compare(object x, object y)
    {
        int yday = 0;
        int xday = ((EventInfo)x).Date.DayOfYear;

        if(y is EventInfo)
        {
            yday = ((EventInfo)y).Date.DayOfYear;
        }
        else if(y is DateTime)
        {
            yday = ((DateTime)y).DayOfYear;
        }

        return xday.CompareTo(yday);
    }
}
```

Our implementation will compare the date field of an EventInfo against a second EventInfo object or a DateTime object (this will prove useful later). Converting dates to a day of the year allows us to ignore the hours, minutes, and seconds of the date, and since the stored procedure only returns events from two consecutive months, we do not have to worry about the year as well.

Once we have the day of the year for both dates, we can delegate to the CompareTo method of the Integer class. A return value less than 0 means x is less than y, that of 0 means x is equal to y, and a return value greater than 0 means x is greater than y.

With a sorted ArrayList of event information, we can mark up the calendar in the OnDayRender event handler. The Calendar control will invoke this event for each day it displays:

```
void EventsCalendar_DayRender(object sender,
                             DayRenderEventArgs e)
{
    e.Day.IsSelectable = false;

    if(!e.Day.IsOtherMonth)
    {
        EventInfo eventInfo = null;
        int index = events.BinarySearch(e.Day.Date,
                                      eventComparer);

        if(index >= 0)
        {
            e.Cell.Controls.Add(
                new LiteralControl("&nbsp;"));
        }

        while(index >= 0)
        {
            MarkCalendar((EventInfo)events[index], e);

            events.RemoveAt(index);
            index = events.BinarySearch(e.Day.Date,
                                      eventComparer);
        }
    }
}
```

First, since we are using the calendar to display information and not select a date, we turn off the selection ability for every cell. Next, we want to skip displaying information in the **filler** days, which are days a `Calendar` control will display to make the control look symmetrical—the padding before and after the days of the month to display.

We perform a binary search to see if there is an event present for the current day being rendered, which requires a sorted `ArrayList`. The `BinarySearch` method does not guarantee that we will find the first `EventInfo` in the `ArrayList` for a given date. The index that `BinarySearch` returns could be the last `EventInfo` object or—if there are multiple events for the day—somewhere in between. A negative number indicates no match.

Instead of searching forward and backward from our current position, we will simply remove the `EventInfo` we found and retry the binary search to see if we are finished. Any time we find a match, we invoke the `MarkCalendar` method shown here:

```
void MarkCalendar(EventInfo eventInfo,
                  DayRenderEventArgs e)
{
    HyperLink link = new HyperLink();
    link.NavigateUrl =
        String.Format("{0}/Events/{1}.aspx",
                      CommunityGlobals.AppPath,
                      eventInfo.ContentPageID);

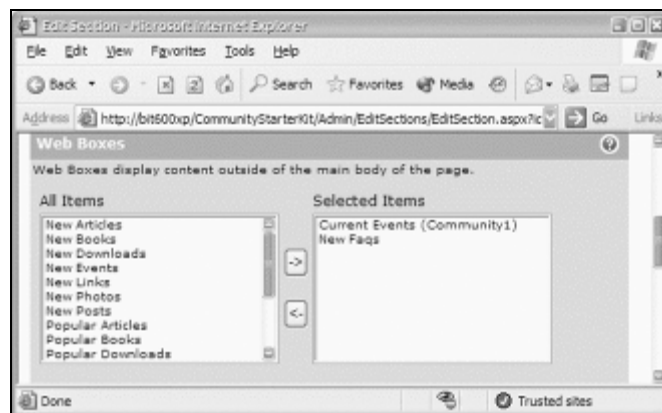
    link.Text = "*";

    link.ToolTip = String.Format(
        "Title: {0} " +
        "Location: {1} " +
        "Speaker: {2}",
        eventInfo.Title,
        eventInfo.Location,
        eventInfo.Speaker);

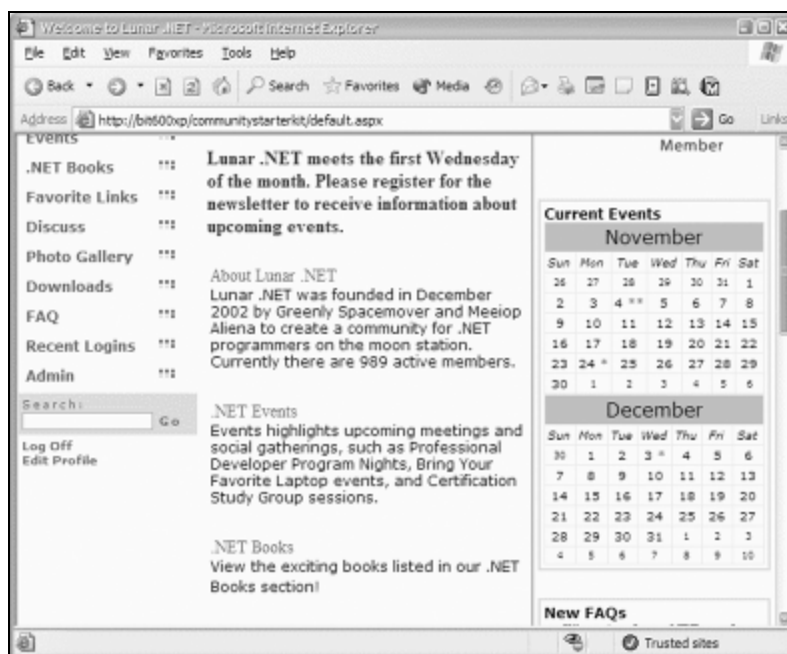
    e.Cell.Controls.Add(link);
}
```

In `MarkCalendar`, we build a hyperlink for the user to click and see the complete details of the event. The URL we build must be an absolute URL since the Current Events Web Box can display from several different areas of the community site, and we want to keep the control cached even if the user has navigated down to a subdirectory. The text we display is very short (one character; you might consider adding a nice icon), but the user can hover over the link to see more information in a tooltip.

If you have placed this Web Box in the proper directory, it should display in the section's edit page as a community-specific Web Box. A community-specific Web Box has the community name in parentheses after the Web Box name:



The final product should resemble the following image:



Summary

In this chapter, we have looked at customizing the community website through the use of user control files. Custom modules display in the main content window and are easy to write and maintain even though they do not come with all the integrated features of a fully featured module.

Web Boxes also use ASCX files to display content around the primary window, and can be community specific or common to all communities. Together these controls can let you fine-tune the customization of your site to fit exactly what your users need.

10

Adding an RSS Feed

A nice feature in the CSK is the ability to add content to a community via a Web Service Box. The Web Service Box can retrieve content from other communities through a web service call, or from a **Really Simple Syndication (RSS)** document retrieved over the network. However, the CSK did not ship with the ability to expose content through RSS. The RSS feed is an attractive option to many community members who would like to know when new content arrives on the site without browsing to the site or checking e-mail. Instead, the user's news aggregator software will periodically scan an RSS feed to retrieve content and alert the user when new items are available.

With RSS and weblogs (blogs) growing in popularity, RSS would be a good feature to have in the CSK. In this chapter, we will take a look at implementing an RSS 2.0 feed. We will need to make some changes to the core of the CSK—the `HttpModule`—and also develop logic to generate the XML for an RSS feed from an ASPX page. We will also see how the CSK architecture allows us to put cross-module cutting functionality in place with a small amount of code. When we are done, we can expose any section module content with an RSS feed.

Before we look at implementing the RSS page, let's learn more about the RSS 2.0 specification we need to abide by.

The RSS 2.0 Specification

The RSS specification, in keeping with the spirit of simplicity, is short and concise. A valid RSS file is also a valid XML 1.0 document, and the specification lists all the optional and required elements (the count of required elements is relatively small). By reading the RSS 2.0 specification in its entirety at <http://blogs.law.harvard.edu/tech/rss>, one can see the ultimate goal of RSS is to make writing software for consuming and producing RSS as easy as possible. Simplicity is one of the keys to the widespread success of RSS. What follows is a sample RSS feed from the community starter kit:

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
  <channel>
    <title>.NET Events</title>
    <link>http://server/CSK/Events/default.aspx</link>
    <description>Events highlights upcoming meetings and social gatherings,
such as Professional Developer Program Nights, Bring Your Favorite Laptop
events, and Certification Study Group sessions.
    </description>
```

```
<item>
  <title>Bring Your Favorite Laptop </title>
  <link>http://server/CSK/Events/42.aspx</link>
  <description>Hands-On Event!
Saturday, June 8, 2003 1200 - 1700 CMST, Building 315, Microsoft Lunar Campus.
This B.Y.F.L is in C#.
</description>
</item>
</channel>
</rss>
```

The top `rss` element and version attribute are mandatory. The `rss` element must contain a single channel element, which in turn requires the `title`, `link`, and `description` elements to describe the service you are providing. A channel may also contain `item` tags to describe the individual content pieces available from the service. The items can be articles, events, diary entries, or any other form of content you could imagine on a web page.

Each RSS item must contain a `title` or `description` element, while `link` is an optional element with a URL to retrieve the rest of the content. There are many optional elements specified for you to augment the information in RSS—for example, a `pubDate` element indicates when an item was published. Our implementation will produce an RSS document similar to the one we just saw.

Community RSS Design

To implement the RSS feed we will follow the design of the community web service. As covered earlier in the book, a single page, `communityService.aspx`, contains the community web service logic. The `HttpModule` will populate a `SectionInfo` object based on the request path and rewrite the request path to point to the service page. Consider:

```
http://server/CommunityStarterKit/Events/communityService.aspx
```

And:

```
http://server/CommunityStarterKit/Downloads/communityService.aspx
```

Both these requests arrive at the same ASMX file, but with different `SectionInfo` objects in the `Context` collection. The web service then retrieves the `SectionInfo` object to produce the appropriate response.

Also there are two classes in the `communityService.aspx` code-behind file that we can utilize:

- `ServiceResponseInfo`
- `ServiceResponseInfoItem`

The first class represents the response we send to a service, and has member properties—such as `ServiceTitle` and `ServiceDescription`—that can be used to populate the sub-elements of the channel element. An `Items` property contains a collection of `ServiceResponseInfoItem` objects. Each object represents a content entry with `Title`, `Link`, and `Description` properties. We can use properties to populate the sub-elements of item.

We can also use the `Community_ServicesGetResponseItems` stored procedure to populate the service response objects. This stored procedure takes a community ID and section ID as

parameters and returns all the fields we require. Before we jump into the code-behind for our RSS feed page, let's first take a look at modifying the `HttpModule`.

HttpModule Changes and Configuration

The first step is to provide a configurable value with the name of the RSS ASPX page. We can achieve this by placing the following new entry in the top-level `web.config` file of the CSK:

```
<pagePaths>
  <add key="basePage"
        value="/communityDefault.aspx" />
  <add key="baseService"
        value="/communityService.aspx" />
  <add key="baseRssFeed"
        value="/communityRSS.aspx" />
</pagePaths>
```

For this exercise, we will name our page `communityRSS.aspx`. In the `web.config` excerpt that we just saw, the path and filename of some of the special CSK pages appear under the `pagePaths` element in `web.config`. The `CommunityGlobals` class manages these entries. The static properties of `CommunityGlobals` allow the rest of the CSK software to access configuration items without knowing all of the details of how the items are stored. This level of indirection is a good practice, which we will follow by adding the `UrlBaseRssFeed` property to `CommunityGlobals`:

```
public static string UrlBaseRssFeed
{
    get
    {
        NameValueCollection nvc;
        nvc = (NameValueCollection)
            ConfigurationSettings.GetConfig(
                "communityStarterKit/pagePaths");

        return AppPath + nvc[ "baseRssFeed" ];
    }
}
```

With the configuration in place, we simply need to modify the `CommunitiesModule` `HttpHandler` in `Engine\Framework\BaseClass\CommunitiesModule.cs` to rewrite the path for incoming RSS requests. The new code should go after the check for a web service request:

```
// Is this a Web Service request?
...

// IS this an RSS request?
if(requestPath.ToUpper().EndsWith("RSS.ASPX"))
{
    Context.RewritePath(
        CommunityGlobals.UrlBaseRssFeed +
        Context.Request.PathInfo);

    return;
}

// Figure out the page and add to Context
...
```

Any request ending with `RSS.aspx` (such as `communityRSS.aspx` or `GetRSS.aspx`) will now end up at our community-wide RSS page.

CommunityRSS.aspx

The actual `CommunityRSS.aspx` file in our implementation is empty except for the `@Page` directive. There are no web controls on the page. Instead, we need to build an XML response and write the XML to the output stream from the code-behind. These high-level steps are reflected in the following `Page_Load` method:

```
SectionInfo sectionInfo = null;
XmlDocument xmlDoc;
ServiceResponseInfo responseInfo = null;

private void Page_Load(object sender, EventArgs e)
{
    Response.ContentType = @"text/xml";

    CreateRss();

    xmlDoc.Save(Response.Output);
}
```

The only method call in the `Page_Load` method is to `CreateRss`. When this method completes, we should have our `XmlDocument` member variable filled with RSS 2.0-compliant XML. Before arriving here, the CSK's `HttpModule` *determined* the section the request is looking for and *created* a `SectionInfo` object. We need to fetch the `SectionInfo` object from the current context in order to build our response. We also want to cache our response to save database queries and processing time. The key into the cache will be based on the section ID.

```
private void CreateRss()
{
    sectionInfo =
        (SectionInfo)Context.Items["SectionInfo"];

    string key = sectionInfo.ID.ToString() + "_RSS";

    xmlDoc = (XmlDocument)Cache[key];

    if(xmlDoc == null)
    {
        xmlDoc = new XmlDocument();
        GetResponseInfo();
        xmlDoc.AppendChild(CreateRssTag());
        Cache.Insert(key, xmlDoc, null,
            DateTime.Now.AddMinutes(60),
            TimeSpan.Zero);
    }
}
```

If the response we need is not in the cache, we need to build the XML from scratch. Most of the work happens in two method calls. Our first method call, `GetResponseInfo`, will retrieve enough information to populate the `ServiceResponseInfo` object. The second method call, `CreateRssTag`, will use the `ServiceResponseInfo` object to build the XML. We will then keep the XML in the cache for 60 minutes so as to avoid extra processing and database queries for the same RSS feed. Let's take a detailed look at `GetResponseInfo`:

```
private void GetResponseInfo()
{
    responseInfo = new ServiceResponseInfo();
    if(!sectionInfo.IsSectionWebService)
```

```

{
    responseInfo.ServiceTitle = "Unpublished";
    responseInfo.ServiceDescription =
        "Section is not published as a service.";
}
else
{
    responseInfo.ServiceTitle =
        sectionInfo.Title;
    responseInfo.ServiceDescription =
        sectionInfo.Description;
    responseInfo.Items =
        GetResponseItems(sectionInfo.ID);
    responseInfo.ServiceLink =
        CommunityGlobals.ResolveAbsoluteUrl(
            sectionInfo.Path);
}
}

```

Notice we have a check to ensure that the section is published. An administrator can control this flag in the application by selecting a checkbox in the Web Services area on the Edit Section page. We can pull all of the properties directly from the `SectionInfo` object, with the exception of the `ServiceLink`. The `ServiceLink` would typically contain the URL for the service end-point. For the web service response, this URL would point to the `.asmx` file. For RSS, this property will populate the link element for the channel. The channel link should contain the URL corresponding to the HTML website for the channel. So, we will compute the URL to the ASPX page listing the section content using `ResolveAbsoluteUrl` from `CommunityGlobals`.

If the section is published, we need to populate the `Items` collection from the database using `GetResponseItems`:

```

public ArrayList GetResponseItems(int sectionID)
{
    ArrayList items = new ArrayList();

    SqlConnection conPortal =
        new SqlConnection(
            CommunityGlobals.ConnectionString);
    SqlCommand cmd =
        new SqlCommand(
            "Community_ServicesGetResponseItems",
            conPortal);
    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.Add("@CommunityID",
        CommunityGlobals.CommunityID);
    cmd.Parameters.Add("@SectionID", sectionID);

    conPortal.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    while (dr.Read())
    {
        ServiceResponseInfoItem responseItem =
            new ServiceResponseInfoItem();
        responseItem.Title =
            (string)dr["ContentPage_Title"];
        responseItem.Link =
            ContentPageUtility.CalculateFullContentPath(
                sectionID, (int)dr["ContentPage_ID"]);
        responseItem.Description =
            (string)dr["ContentPage_Description"];
        items.Add(responseItem);
    }
}

```

```
        conPortal.Close();  
        return items;  
    }
```

The `GetResponseItems` method is fairly typical of the CSK data-access code. For some reason, however, the CSK designers did not define a constructor for `ServiceResponseInfoItem`, which takes a `SqlDataReader` reference. We could add a constructor ourselves or set the properties explicitly (as has been done here). We add each item to an `ArrayList` collection to return to the caller.

The next significant method call is to `CreateRssTag`. This method creates the top document element for the XML response and must include the RSS version:

```
private XmlElement CreateRssTag()  
{  
    XmlElement e = CreateElement("rss");  
    e.Attributes.Append(CreateAttribute("version",  
                                        "2.0"));  
    e.AppendChild(CreateChannelTag());  
    return e;  
}
```

Our class also has two helper methods to work with XML. `CreateElement` and `CreateAttribute` create `XmlElement` and `XmlAttribute` objects respectively by forwarding calls to the `XmlDocument` object's methods with the same names:

```
private XmlElement CreateElement(string name)  
{  
    return CreateElement(name, null);  
}  
  
private XmlElement CreateElement(string name, string data)  
{  
    XmlElement e = xmlDoc.CreateElement(name);  
    if(data != null)  
    {  
        e.InnerText = data;  
    }  
    return e;  
}  
  
private XmlAttribute CreateAttribute(string name, string data)  
{  
    XmlAttribute a = xmlDoc.CreateAttribute(name);  
    a.Value = data;  
    return a;  
}
```

Once we have added the `rss` tag to the document, we can add the `channel` tag with another method call:

```
private XmlElement CreateChannelTag()  
{  
    XmlElement e = CreateElement("channel");  
    e.AppendChild(CreateElement("title",  
                                responseInfo.ServiceTitle));  
    e.AppendChild(CreateElement("link",  
                                responseInfo.ServiceLink));  
    e.AppendChild(CreateElement("description",  
                                responseInfo.ServiceDescription));  
}
```

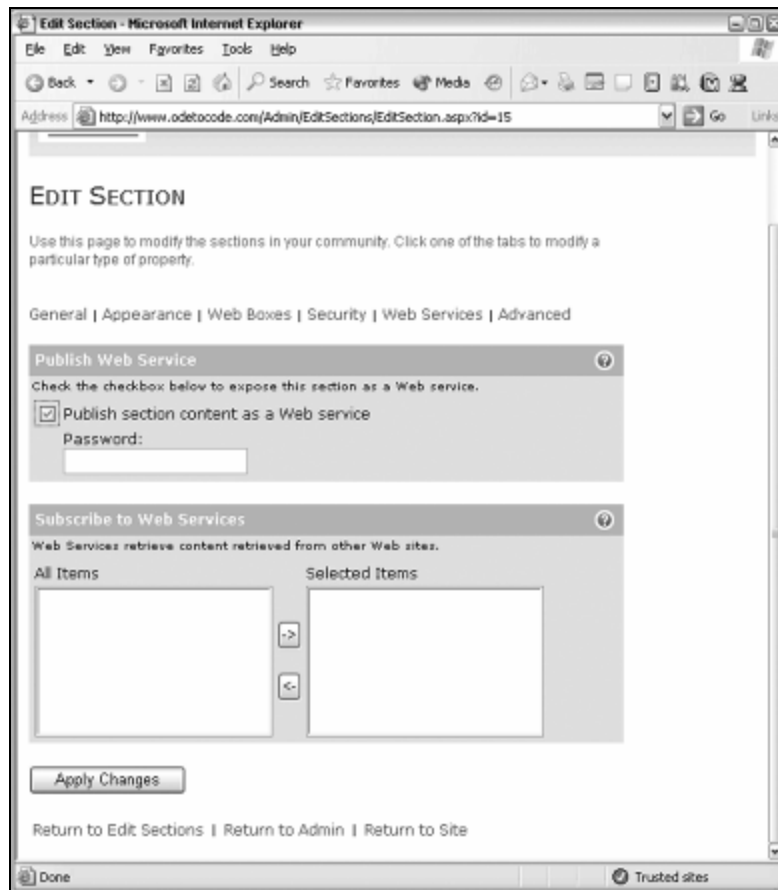
```
        foreach(ServiceResponseInfoItem i in
                responseInfo.Items)
        {
            e.AppendChild(CreateItem(i));
        }
        return e;
    }
```

This method creates all of the required elements for the `channel` tag by pulling properties from the `ServiceResponseInfo` object. As the final step, we need to loop through the objects in the `Items` property and create an `item` tag for each object with the `CreateItem` method:

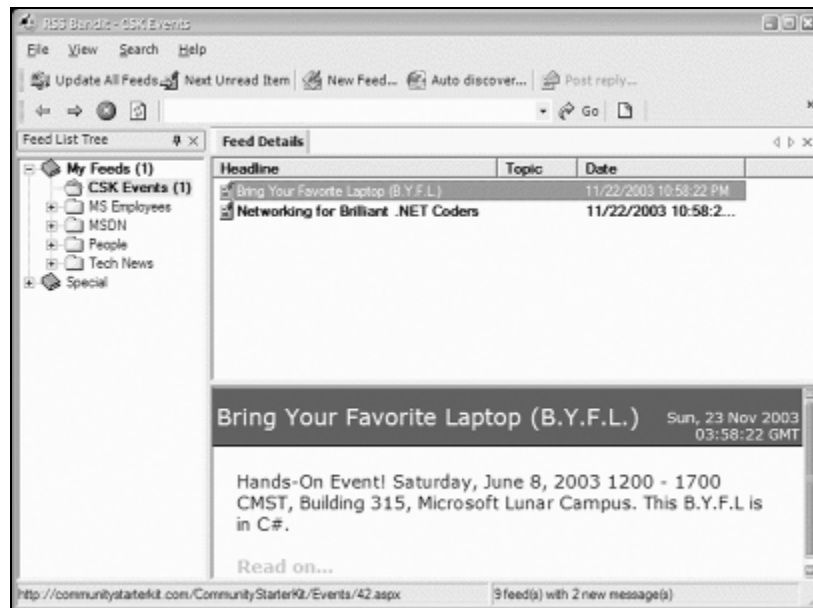
```
private XmlElement CreateItem(ServiceResponseInfoItem
                             responseInfoItem)
{
    XmlElement item = CreateElement("item");
    item.AppendChild(CreateElement("title",
                                   responseInfoItem.Title));
    item.AppendChild(CreateElement("link",
                                   responseInfoItem.Link));
    item.AppendChild(CreateElement("description",
                                   responseInfoItem.Description));
    return item;
}
```

Testing the RSS Feed

To test a feed, we log into a community site in an administrator role and select a section to be published as a web service.



There are two ways to test the created RSS feed. The first method involves using a news aggregator that displays RSS content on the desktop. The following screenshot is an example of reading the Events section of the default CSK using RSS Bandit. RSS Bandit is a free news aggregator written in .NET and available from <http://www.rssbandit.org/>. The RSS feed for the Events section would be <http://server/CommunityStartKit/Events/communityRSS.aspx>.



A second way to test the RSS feed is to use a validator. One such validator is available at <http://feedvalidator.org>. Simply enter the URL to your feed, and the feed validator will inspect the XML to see if it meets the RSS 2.0 specification. If you are working on a server that is not on the network and have a Python interpreter installed, you can download the sourcecode for the validator to run locally. Validation is a good step to take if you modify or add to the RSS feed.

Advertising the RSS Feed

As a final step, we need to provide a link so that users find the RSS feed. Since any section of the community website could be publishable, we do not want to modify each module to display the RSS feed link. Instead, we will modify a common control, the `SectionTitle` control, which displays on every list of section content.

The `SectionTitle` control lives in `SectionTitle.cs` under the `Controls` directory of `Engine\Framework\ContentPages`. The control currently takes the `Title` property of the current `SectionInfo` object in `Context` and writes the title out during the `RenderContents` method. A slight modification will also add a hyperlink and an image representing our RSS feed:

```
public SectionTitle() : base() {
    // assign default css class
    CssClass = "sectionTitle";

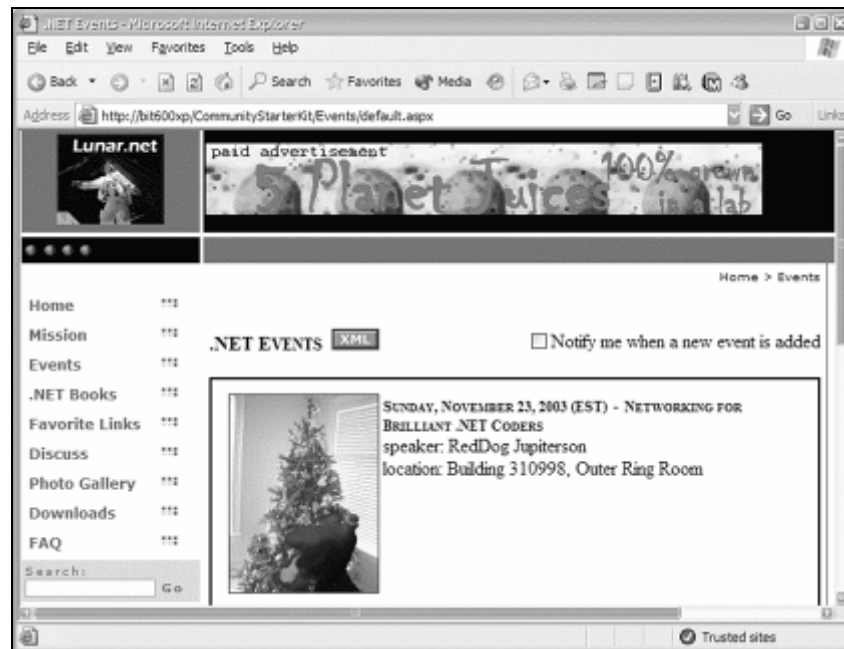
    // Get SectionInfo object
    if (Context != null) {
        SectionInfo objSectionInfo =
            (SectionInfo)Context.Items["SectionInfo"];
        _text = objSectionInfo.Title;
    }
}
```

```

        if(objSectionInfo.IsSectionWebService)
        {
            _text += String.Format(
                "&nbsp; <a href='{0}'><img src='{1}'" +
                "alt='RSS Feed'></a>",
                "communityRSS.ASPX",
                CommunityGlobals.AppPath +
                "/communities/common/images/xml.gif");
        }
    }
}

```

This code assumes that there is an `xml.gif` file in the `/communities/common/images` directory. You can download the small XML icon from the page with the RSS specification (<http://blogs.law.harvard.edu/tech/rss>). The code needs to build a complete URL to the *image* because we do not know all the paths where a `SectionTitle` might show up. In contrast, the hyperlink to the *feed* itself can be a relative path as long as the page ends with `RSS.aspx`. As we saw earlier, the `HttpModule` detects the request and takes care of the rest. The new feature can be seen in the following screenshot:



Summary

In this chapter, we have taken a look at implanting an RSS feed for the CSK. This included a small change in the core driver of the CSK—the `CommunityModule HttpModule`. The ability to add RSS to existing and new modules with such a small change is a testament to the flexibility and architecture of the CSK as a whole.

11

Deployment

In this chapter, we will cover several strategies for deploying the CSK. Although deployment can happen through a wide variety of methods, there are four general steps to take in order to have your CSK site up and running:

- Installing content files
- Configuring IIS
- Setting up the database
- Community configuration

The chapter assumes that you are building a custom version of the CSK and thereby foregoing the packaged installation for the CSK. We will talk about strategies for deploying multiple communities on a single server, as well as configuration differences between hosting on the Windows 2000 Server platform versus Windows Server 2003. By the end of the chapter, you will be familiar with the techniques and tips required to get your own CSK site up and running.

Installing Content Files

There are a number of options for getting your content files onto a production server. Content files include ASPX, ASCX, and ASMX files, any configuration files, and the compiled output of the project (`ASPNET.StarterKit.Communities.dll`). In short, all of the files that are required for the CSK to execute successfully on the web server. You do not need to copy any of the source code (CS files) as these are compiled into the project's DLL.

Make sure you deploy a release mode build of the CSK. A release mode build will run more efficiently and use fewer resources on the server. To make sure you are building in release mode, right-click on the CSK solution in the Solution Explorer window and select Properties. In the dialog that appears, go to the Configuration Properties node and select the Release configuration in the dropdown next to the community project name.

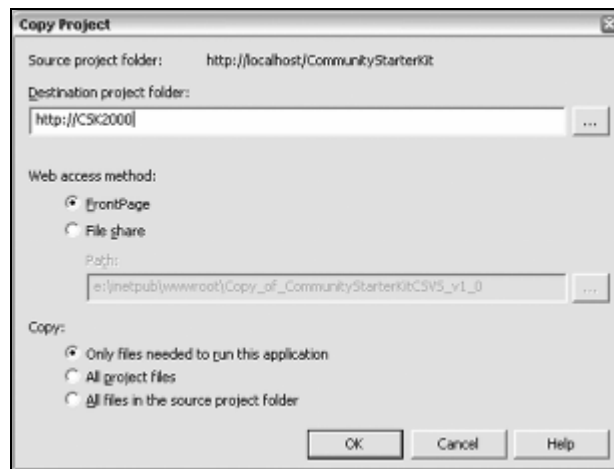
Once the project is built, you are ready to move content to the destination server. This can be done using two techniques.

Copy Project

This technique uses the Copy Project command from Visual Studio .NET. It is the best suited deployment model for either of the following circumstances:

- The destination server has FrontPage Server Extensions (FPSE) installed. Many web hosting companies allow remote authoring and management of a site over the Internet via FPSE.
- The destination server is on your local network.

In both circumstances, you will need appropriate access rights to the destination and you may need to provide a username and password to gain access. To begin, select the Copy Project item from the Project menu to bring up the dialog as shown:



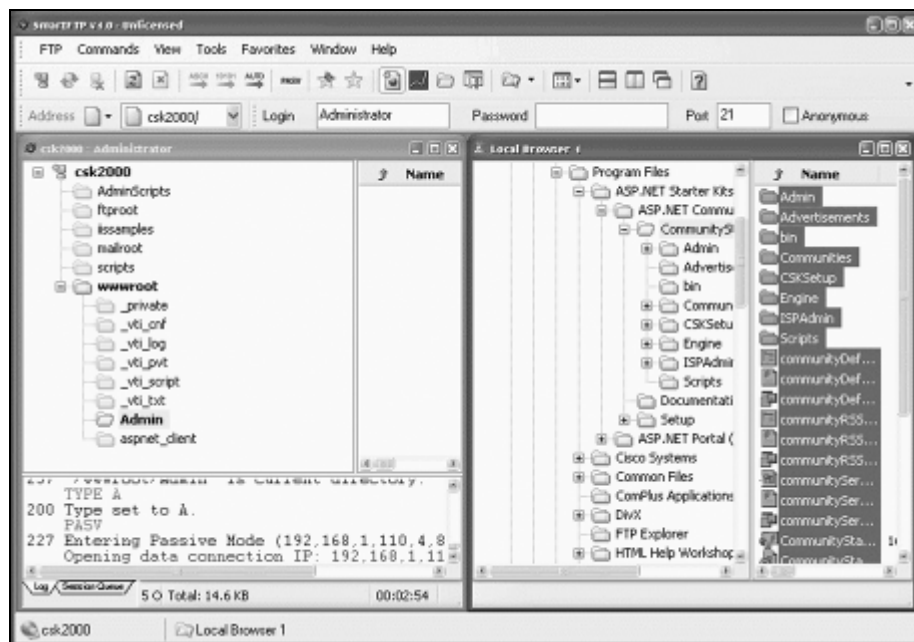
Using the FrontPage access method, you point Visual Studio to the destination for the project using an HTTP URL. For File share access, you can provide a path in UNC format, or click the ellipsis button to browse to the destination. For the Copy option, you will typically want to select Only files needed to run this application. Click the OK button to have the IDE transfer the content and binary files to the server.

Copy Project is the easiest technique to deliver files to production. There will still be some web server configuration to take care of before the CSK is functional, but before we move to the configuration section let's take a look at another of deployment technique.

Xcopy Deployment

Xcopy deployment is a generic term in .NET for deploying an application by copying files. You can copy files using Windows Explorer, File Transfer Protocol (FTP), or the XCOPY command from a DOS prompt. Since the CSK requires no assemblies in the global assembly cache (which requires a registration step on the server) this is a viable alternative. However, there is some additional IIS configuration required which we will cover in the next section.

Xcopy is the best choice when your hosting provider gives you access via FTP instead of using FPSE. There are many commercial, shareware, and free FTP clients available for download on the Internet. Make sure you choose an FTP client that allows you to recursively transfer multiple files and directories easily. One example of such a client is SmartFTP from www.smartftp.com. SmartFTP is free for students and for non-commercial use. In the following screenshot, we are selecting all the files and directories in the root directory of the CSK project to drag, drop, and upload to a web server. The client program will copy all selected items to the server, creating directories if need be.



Note that we are copying all files from the CSK directories (and not just the binary and content files). If you are hesitant to copy the source code and other information to the web server, then you may consider setting up a staging area on the development machine. The following batch file assumes there is a stage directory under the CSK website. It will copy all files into the stage directory, and then remove the files you do not need to deploy. You can then FTP the content from the staging directory.

```
>xcopy /s *.* ..\stage
cd ..\stage
del /s *.cs
del /s *.resx
del /s *.bat
del /s *.txt
del /s *.pdb
del /s *.csproj*
del /s *.sln
del /s *.vsscc
```

Even if your source code does make it to the server, ASP .NET catches requests for cs, vb, csproj, resx, config, and other file types that a malicious user might want to see. It refuses to return them and instead, routes the request to `System.Web.HttpForbiddenHandler`. You can see the restricted file types in the `httpHandlers` section of `machine.config`.

Configuring ASP.NET

Configuring ASP.NET on the server will depend on the version of Windows being used. Windows 2000 was a pre-.NET operating system and you may need to install a couple of items before you can get started. Windows 2003 ships with the ASP.NET runtime available but it's disabled by default. We will take a look at getting ASP.NET started on both these platforms. We will assume you are going to deploy .NET version 1.1.

ASP.NET on Windows 2000

To run ASP.NET on Windows 2000, you'll need to apply at least service pack 2. You'll also need the latest MDAC (2.7), as Windows 2000 with service pack 4 applied has only MDAC 2.53. You can download the latest MDAC version from www.microsoft.com/downloads.

If IIS was already installed on Windows 2000 when you ran the .NET Framework setup, ASP.NET should be ready to go. If you installed IIS after installing the .NET Framework, or if you uninstalled and reinstall IIS after the .NET setup, you'll need to use the ASP.NET IIS registration tool (`aspnet_regiis.exe`). This tool is found in the `bin` directory of the .NET runtime installation, and you can invoke it from the command prompt as

```
aspnet_regiis -i
```

This tool installs the correct script mappings to process ASP.NET files, and also places client-side script files (used by validation components) into the proper directory.

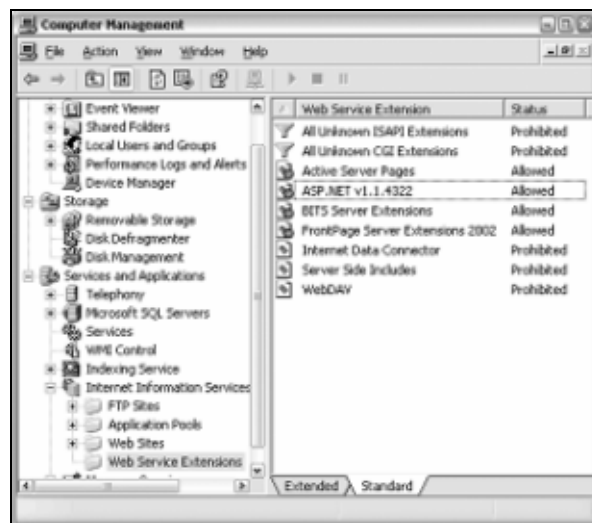
If you are installing ASP.NET on a domain controller, you'll need to take additional steps outlined in the Microsoft knowledge base article Q315158 available at support.microsoft.com.

ASP.NET on Windows 2003

The .NET Framework version 1.1 including ASP.NET is included in Windows Server 2003 editions. However, due to the heightened security awareness in the operating system, ASP.NET processing starts in a disabled state. Windows setup does not install IIS by default; you'll need to select IIS as an option during server setup if you are building your CSK host from scratch.

If you are working with Windows Server 2003 Web Edition, register ASP.NET as an allowable web extension. You can do this with the ASP.NET registration tool in the same manner as with Windows 2000.

In all versions of Windows Server 2003, you'll need to open the IIS management console and navigate to Web Service Extensions node. Select ASP.NET and click Allow. To allow deployment via the Visual Studio Copy Project command, you need to enable FrontPage Server Extensions.



CSK Site Configuration

The next step in deployment is to configure the CSK website. You can deploy the CSK into the root of a website or into a virtual directory; the settings shown in this section will remain the same for both scenarios. If you used the Copy Project command, some of the following settings will already be in place for you, but we will review all pertinent configuration items to ensure a working CSK. Right-click on the website or virtual directory (create one first if need be) that you wish to host your communities and select Properties.



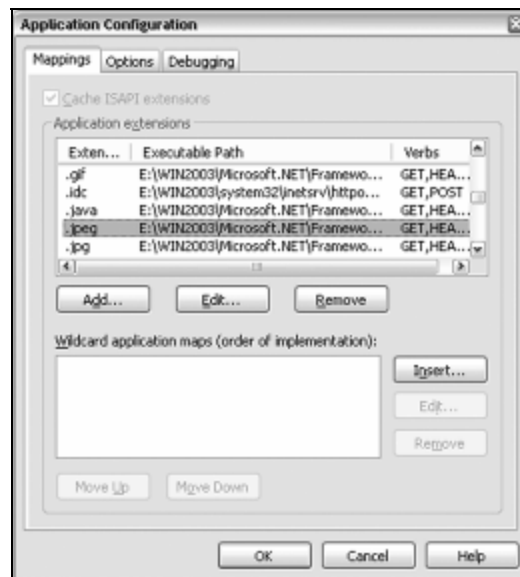
If you are working at the web-site level, go to the Home Directory tab; else go to the Virtual Directory tab. Both should resemble the previous screenshot.

First, make sure the Local path setting points to the root directory of the CSK (the directory containing `communityDefault.aspx`). Make sure the Read checkbox is selected. This will allow users to read and download files in this directory.

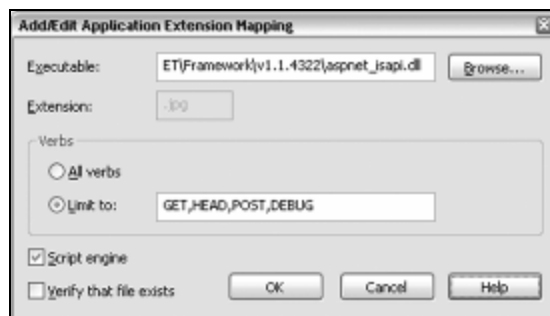
You can have the option to check Log Visits. This instructs IIS to record visits to this application in a log file. Specific logging properties can be tweaked by right-clicking and selecting properties on the website in the MMC.

Make sure the CSK is configured as a web application. If the button next to the Application Name textbox says Create, click Create to construct an application. If the button says Remove (as shown in the preceding screenshot), the site has already been configured as an application.

Now, remember that the CSK sometimes needs to serve image requests from binary fields in the database. For these images to work, certain application extension mappings must be in place. Click on the Configuration button to see the current mappings:



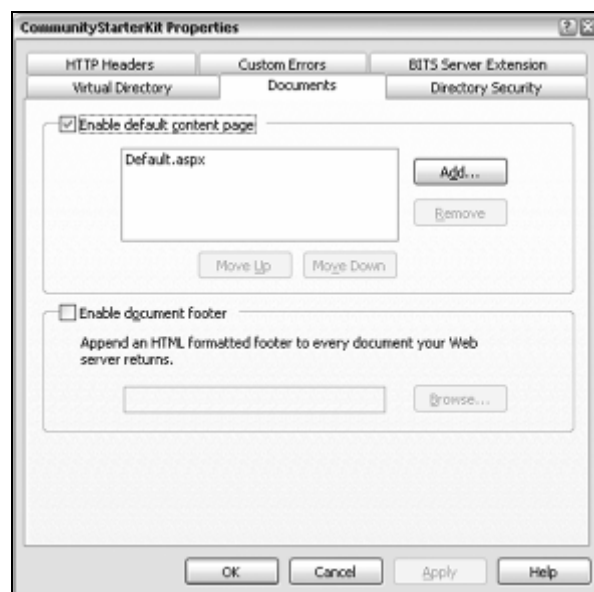
We will need to add .gif, .jpeg, and .jpg mappings. Click on the Add button to bring up the Add/Edit Application Extension Mapping dialog as shown below. For each image extension that you add, make sure the Executable textbox points to `aspnet_isapi.dll`:



It is also important to uncheck the Verify that file exists checkbox. If you do not uncheck this setting, IIS will first check to see if the image file exists before giving control to the application extension. Since many images in the CSK do not exist on disk but in the database, this can be a cause for missing images at run time.

Close the Application Configuration dialog and return to the Properties dialog for the site or virtual directory containing the CSK.

Lastly, you need to set up a default document. Go to the Documents tab, and make sure Enable default content page is selected. You will also want to move Default.aspx to the top of the listbox control. If Default.aspx is not present, add the file name and move it to the top of the list. You can delete the other entries in this listbox control:



A quick note if you are hosting the CSK in the default website of Windows 2000: Windows 2000 defaults to having a `Scripts` virtual directory under the default website. This virtual directory will point to the `Inetpub\Scripts` directory.

Unfortunately, the CSK also contains a `Scripts` directory. The CSK `Scripts` directory contains, among other files, the `.js` file for the HTML textbox. The `Scripts` virtual directory set up by Windows will hide the CSK `Script` directory and none of the `.js` files will be available to the CSK. As a result, you won't see the HTML textboxes appear. The easiest solution to this problem is to delete or rename the `Scripts` virtual directory set up by Windows.

If you are working with a hosting provider, chances are you will not have access to the management console for IIS. Instead, you'll have to work with the provider to have your website properly configured.

Fortunately, none of the settings we have covered here are extraordinary. Any competent ISP should be able to make any necessary changes. In fact, most hosting providers will already have all of the settings except for the script mappings in place. With the IIS settings properly configured for the CSK, it is time to move our database to the production server.

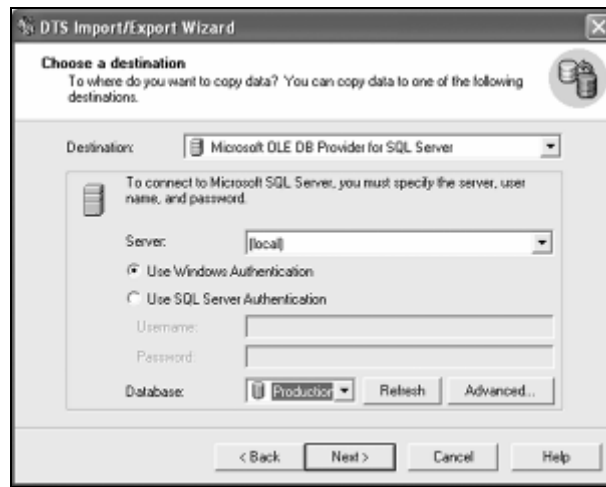
Database Deployment

To deploy the CSK you will need to create a database with all of the necessary objects (tables, stored procedures, and so on) for your production website. In addition, you may want to copy the records and data stored in your development database to the server. In this section we will examine how to achieve this by using Data Transformation Services (DTS), using Backup / Restore features of SQL Server, and using just SQL script.

DTS Deployment

In this DTS walkthrough, we will create a new database on our production server for the CSK and move all objects and data from our development database. To begin, start the SQL Server Enterprise Manager. Right-click the development database, select **All Tasks**, and select **Export Data**. This should bring up the DTS Import/Export Wizard.

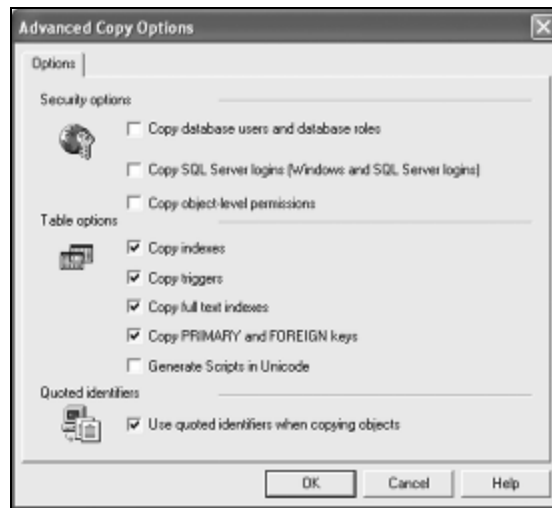
In the **Choose a Data Source** step of the DTS wizard, your development database should already be selected, so click **Next**. In **Choose a destination**, enter the connection parameters to connect to the production database server. If you already have a database configured, you can select the database in the **Database** drop-down list. Otherwise, select **new** in the dropdown and enter a name for the CSK database, then select **Next**.



In the next screen, select Copy objects and data between SQL Server databases and click Next.



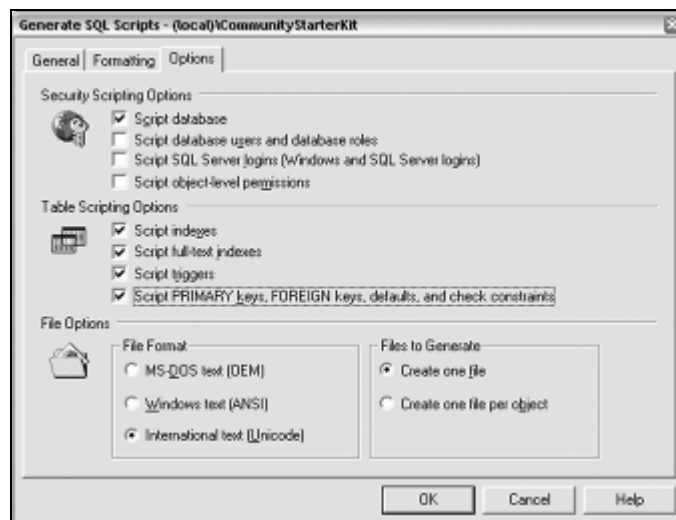
In the next screen (Select Object to Copy), uncheck the Use default options checkbox and click the Options button. In the screen that appears (shown next), uncheck all three of the Security options checkboxes. After this point, you can accept all the default wizard settings. On clicking Finish, the DTS job should begin copying objects and data to your production server:



Script Deployment

If you do not have connectivity from your development environment to the production environment, you may be faced with the task of producing scripts for deployment. DTS has the capability to script a database including all objects. However, you will still need a way to move data to the production server. Let's first take a look at scripting the database.

Highlight and right-click the development database in SQL Enterprise Manager, select All Tasks, and select Generate SQL Script. In the resulting dialog, click the Show All button and check Script all objects. This should result in placing all of the CSK objects into the Objects to be scripted list view. Next, click on the Options tab and select the options as shown in the following screenshot:



After you click OK, you'll be asked for a location and file name for the SQL script to be saved.

You can now take the SQL Server script created by Enterprise Manager to the production server, and execute in Query Analyzer or from the `osql` command-line tool. The script will create the CSK database and schema. After you replace the parameters in angled brackets with the setting for your environment, the `osql` command would look like the following command line:

```
osql -S <servername> -d <databasename> -U <username> -P <password> -i filename>
```

Unfortunately, moving data is not quite as easy. There are several options, including using DTS to export data into comma-delimited files to take to the server and import. Unfortunately this works on a table-by-table basis making it quite cumbersome. Another option is to buy a tool such as SQLDiff (<http://www.lockwoodtech.com/sqldiff.htm>) to generate SQL scripts of your data, or write a tool to do the same.

Backup and Restore

An easier alternative is to use the SQL Server backup and restore tools. In Enterprise Manager, right-click on the development database and choose All Tasks then Backup Database. Click the Add button and select a filename and location for the backup. Select Overwrite existing media and click OK. You should now have a file that will completely recreate your development database on the server.

To restore the backup in production, right-click on the Databases node in Enterprise Manager, select All Tasks, and then select Restore Database. Type a name for the database in the Restore As Database drop-down box, then select the From device option from the Restore options. Click on Select Devices. In the Choose Restore Devices dialog box, click Add and browse to the backup file you created and copied to the server. Then click OK to return to the previous dialog.

The important step here is to click on the Options tab and verify the physical filenames for the database. Unless your development and production environments are configured the same, you may need to change the drive letter and path to where the database data and log files will reside. Once this is complete, click OK and the restore operation will begin.

Database Ownership Problems

When moving databases between development and production servers with a hosting provider, it is possible to encounter ownership issues on stored procedures and functions. The symptom is in seeing error messages similar to the following:

```
Invalid object name 'dbo.Community_IsContentVisible'.
```

If you run into this problem, Reddnet.net has scripts to reconcile the problems at the following URL: <http://www.reddnet.net/Programming/CSK+Stuff/CSK+Files/CSK+1.0/268.aspx>.

Once the database is in place, the final step is to configure the CSK application itself.

Community Starter Kit Configuration

There are two steps to CSK configuration:

25. Edit `web.config` for a production environment.
26. Configure your communities through the ISP Admin tools.

Web.Config

There are few modifications you need to make to the `web.config` file in production. You may want to change the `IspUsername` and `IspPassword` values to be more secure. Change the `connectionString` to point to your production database.

There are a couple of changes required in the `system.web` section. In the `compilation` element, make sure that the `debug` attribute is `false`. Set the `mode` attribute in `customErrors` to `RemoteOnly`. These settings will ensure that the CSK site runs at peak performance without any error messages revealing source code getting out to users. Even after doing a release build in the CSK, you'll still need to set this `debug` attribute to prevent the ASPX pages from being compiled into debug code.

With ASP.NET 1.1, all requests are scanned for an attempt to POST HTML content. If the runtime detects HTML content in a form POST, it will throw an exception. This feature helps applications avoid script exploits (where a user sends malicious code to the application). This also prevents a CSK admin from uploading HTML content to the site.

Since the CSK already has built-in mechanisms to filter HTML uploading, we can disable this behavior by placing the following element inside `system.web`:

```
<pages validateRequest="false" />
```

ISPAdmin

Log in to the ISPAdmin page (<http://server/ISPAdmin/IspLogin.aspx>) to configure your communities.

If you have only a single community, you can leave the default values in the Edit Community page (* for community domain and * for community subdomain). You can adjust the community name, primary domain, SMTP Server, and database quota to suit your requirements.

If you plan on hosting multiple communities, you'll need to correctly set the domain and subdomain for each community. As an example, let's imagine you want to host the following two communities: `CommunityOne.packtpub.com`, and `CommunityTwo.packtpub.com`. For both communities, the community domain is `packtpub.com`. The subdomains are `CommunityOne` and `CommunityTwo` for the first and second communities respectively. There is no need to set up multiple websites or virtual directories in IIS to host multiple communities. There no need to make copies of the CSK files.

The beauty of the CSK architecture is its ability to serve multiple customized communities from the same website, directories, and database.

Working with Other Sites

On some websites the CSK might not be the only web application. In this section we will give you some tips on running other web applications under the CSK. The CSK assumes total control of the directories under it. If you place other ASP.NET applications, pages, or images, in a subdirectory of the CSK, you may have trouble linking to and executing these items.

Let's assume you have a subdirectory where you will upload images to link to. The CSK, however, intercepts the requests for image files and tries to serve the request by finding the image in the CSK database. This leaves the images in your own subdirectory as broken links in the browser. The solution is to override the top-level `web.config` file of the CSK by placing another `web.config` file in the subdirectory with instructions to remove the image file mapping. The following `web.config` file will override the image mappings for a subdirectory:

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <system.web>
    <httpHandlers>
      <remove verb="*" path="*.jpg" />
      <remove verb="*" path="*.jpeg" />
      <remove verb="*" path="*.gif" />
    </httpHandlers>
  </system.web>
</configuration>
```

There is another scenario to consider while running an entire application from a subdirectory of the CSK. The ASP.NET runtime still reads the top-level CSK `web.config` and sees a request to add an `HttpModule` by the name of `CommunitiesModule`. Unfortunately, the ASP.NET runtime will go looking for an assembly named `ASPNET.StarterKit.Communities.dll` in the `bin` directory of the application living under the CSK. Since this application is not the CSK, the `.dll` will not exist and the runtime throws an exception. You might think that placing the following line in the `web.config` for the application would resolve the problem:

```
<httpModules>
  <remove name="CommunitiesModule" />
</httpModules>
```

Unfortunately, the runtime still looks for the CSK assembly. The only workaround is to copy the `ASPNET.StarterKit.Communities.dll` assembly into the `bin` directories of any web applications you need to run under the CSK.

Summary

We detailed several ways to deploy and configure your CSK site and database. Fortunately, the CSK works well in environments ranging from dedicated servers to shared hosting. Hopefully, you will have an environment where you can use simple tools, such as the Visual Studio .NET IDE and SQL Server Enterprise Manager to move your site into production with little effort. If not, the steps are still outlined in this chapter to make a smooth transition from development to production. In more restrictive environments, you can use FTP clients and SQL scripts to build your CSK site on a remote server.



CSK Controls

BriefDescription

BriefDescription can be found in the following ContentSkins skin files:

- Articles_AddArticle.ascx
- Articles_Article.ascx
- Books_Book.ascx
- Downloads_Download.ascx
- Events_Event.ascx
- Links_AddLink.ascx
- Links_Link.ascx
- PhotoGallery_Photo.ascx

ContentList

The ContentList control is used in the following skin files:

ContentSkins:

- Articles_ArticleSection.ascx
- Books_BookSection.ascx
- Discuss_DiscussSection.ascx
- Downloads_DownloadSection.ascx
- Events_EventSection.ascx
- Events_PastEvents.ascx
- Links_LinksSection.ascx
- Moderation_ModerateSection.ascx
- ParentSection_ParentSection.ascx
- PhotoGallery_PhotoGallerySection.ascx

- Search_Search.ascx
- Topics_Topic.ascx

ControlSkins:

- Comments_DisplayComments.ascx
- Topics_TopicMenu.ascx

TemplateSkins:

- Comments_DisplayComments-ItemTemplate-Embedded.ascx
- Comments_DisplayComments-ItemTemplate-Nested.ascx
- Comments_DisplayComments-ItemTemplate-Threaded.ascx
- Discuss_Comments-ItemTemplate-Embedded.ascx
- Discuss_Comments-ItemTemplate-Nested.ascx
- Discuss_Comments-ItemTemplate-Threaded.ascx

DateCreated

Here are the skin files that use this control:

- Articles_Article.ascx
- Comments_Comment.ascx
- Discuss_Post.ascx
- Downloads_Download.ascx
- PhotoGallery_Photo.ascx

DisplayQuota

This is used by default in two skin files:

- Downloads_AddDownload.ascx
- PhotoGallery_AddPhoto.ascx

HtmlTextBox

HtmlTextBox is used by default in the following skin files:

- Articles_AddArticle.ascx
- Books_AddBook.ascx
- Discuss_AddPost.ascx
- Downloads_AddDownload.ascx

- `Events_AddEvent.ascx`
- `HtmlPage_EditHtmlPage.ascx`
- `PhotoGallery_AddPhoto.ascx`

ItemReadMoreLink

ItemReadMoreLink is used in the following skin files:

- `Articles_ArticleSection.ascx`
- `Books_BookSection.ascx`
- `Downloads_DownloadSection.ascx`
- `Events_EventSection.ascx`
- `Events_PastEvents.ascx`
- `Moderation_ModerateSection.ascx`

ItemTitleLink

ItemTitleLink is used in the following skin files:

ContentSkins:

- `Discuss_DiscussSection.ascx`
- `Search_Search.ascx`
- `Topics_Topic.ascx`

TemplateSkins:

- `Comments_DisplayComments-ItemTemplate-Threaded.ascx`
- `Discuss_Comments-ItemTemplate-Threaded.ascx`

LengthValidator

LengthValidator is used in the following skin files to limit the length of the brief description textbox:

- `Articles_AddArticle.ascx`
- `Books_AddBook.ascx`
- `Events_AddEvent.ascx`
- `Links_AddLink.ascx`
- `PhotoGallery_AddPhoto.ascx`

PageFooter

PageFooter is implemented in the following content skin files:

- Articles_ArticleSection.ascx
- Books_BookSection.ascx
- Discuss_DiscussSection.ascx
- Downloads_DownloadsSection.ascx
- Events_EventSection.ascx
- Events_PastEvents.ascx
- Links_LinksSection.ascx
- ParentSection_ParentSection.ascx
- PhotoGallery_PhotoGallerySection.ascx

SectionTitle

SectionTitle is used in the following skin files (they are all content skins):

- Articles_AddArticle.ascx
- Articles_ArticleSection.ascx
- Books_AddBook.ascx
- Books_BookSection.ascx
- Discuss_AddPost.ascx
- Discuss_DiscussSection.ascx
- Discuss_Post.ascx
- Downloads_DownloadsSection.ascx
- Events_AddEvent.ascx
- Events_EventSection.ascx
- Events_PastEvents.ascx
- Links_LinksSection.ascx
- Moderation_ModerateSection.ascx
- ParentSection_ParentSection.ascx
- PhotoGallery_AddPhoto.ascx
- PhotoGallery_PhotoGallerySection.ascx

Sorter

The Sorter control is used in the following content skin files:

- Articles_ArticleSection.ascx
- Books_BookSection.ascx
- Downloads_DownloadsSection.ascx
- Links_LinksSection.ascx
- PhotoGallery_PhotoGallerySection.ascx

Title

The Title control is used in the following skin files:

- Articles_AddArticle.ascx
- Articles_Article.ascx
- Books_Book.ascx
- Comments_AddComment.ascx
- Comments_Comment.ascx
- Discuss_AddPost.ascx
- Discuss_Post.ascx
- Discuss_ReplyPost.ascx
- Downloads_Download.ascx
- Events_Event.ascx
- PhotoGallery_Photo.ascx

UploadValidator

UploadValidator is used in these skin files:

- Downloads_AddDownload.ascx
- PhotoGallery_AddPhoto.ascx

Index

@sortOrder parameter	170	answer to FAQ, control for displaying	206
<database> element, Web.Config file	127	Application_BeginRequest event handler	179
<httpHandlers> element	133	Application_BeginRequest method, CommunitiesModule class	135
<httpModules> element	133	ArrayList class	179
<pagePaths> element, Web.Config file	127	ArrayList object	68
<services> element, Web.Config file	127	article skin	188
A		ArticleBodyText control	189
AddArticle method	176	ArticleEditContent control	90
AddCommunity subfolder, ISPAdmin folder	133	ArticleInfo object	176
AddFaq class, content class	209	ASP.NET	
AddFaq method, FaqUtility class	204	starter kits	1
adding a new FAQ, stored procedure for	198	ASP.NET installation for Windows 2000	250
adding article	168	ASP.NET installation for Windows 2003	250
adding comments	47	ASPNET.StarterKit.Communities.Faq namespace	207
adding content, AddContent control	89	atomicity of data update	199, 200
adding new sections	51, 115	Author control	91
AddSearchKeys method	177	B	
Admin folder, basic CSK folder	129	BookEditContent control	90
Admin subfolder, Engine folder	134	BriefDescription control	92
administering advertisements	66	C	
administering quotes	67	CacheDependency object	186
AdRotator control, use in advertisement banners	66, 81	CalculateMetaDescription method	177
Advanced tab, Community Administration Menu		CalculateMetaKeys method	177
exposing data	47	cascading referential integrity	198
Advertisements folder, basic CSK folder	131	closing database connections	205

code-behind files	208, 218	Community_ArticlesAddArticle, stored procedure	168
Communities folder, basic CSK folder	131	Community_ArticleServiceSelect, stored procedure	168
Community Administration Menu	37	Community_ArticleServiceUpdate, stored procedure	168
drawbacks	37	Community_ArticlesGetArticle, stored procedure	
view of the tool	39	ordering content	169
Community Administrator role	27	Community_ArticlesGetArticles, stored procedure	Error! Not a valid bookmark in entry on page 169
Community Moderators role	27	Community_Books table	166
Community Starter Kit (CSK)	1	Community_BooksGetBooks, stored procedure	172
account administration	33	Community_ContentPages table	166, 194
administrative interface	1	Community_Faqs table	196
caching data	202	cascading referential integrity restraints, use of	198
commenting the content	28	Community_FaqsAddFaq stored procedure	198
CommunityStarterKit database	8	Community_FaqsEditFaq stored procedure	199
custom controls	88	Community_GetCommentCount, stored procedure	171
deployment strategies	247	Community_GetContentItem function	173
folder structure, root folder	125	Community_GetHasRead, stored procedure	171
functionality	2	Community_GetOrderedContent, stored procedure	169
HTML Editor, built-in	32	Community_GetPagedSortedContent, stored procedure	171, 200
inheriting skins and styles	76	efficient way of writing code	201
installation	3	Community_GetUserID function	168
license agreement	3	Community_maintenanceInitializeFaqs stored procedure	202
main folders	129		
moderating content	27		
module architecture diagram	165		
notifications to members	30		
rating the content	30		
release mode version	247		
security	26		
skinnable interfaces	2		
skins, built-in	76		
styles, built-in	76		
system roles	27		
user quotas	31		
Community_AddContentPage stored procedure	199		
Community_AddContentPage, stored procedure	169		

Community_SearchAddSearchKey, stored procedure	177	ContentList control	93–95
Community_SearchKeys stored procedure	177	DataGrid/DataList compared	95
Community_SearchNoiseWords table	177	used in new website creation	120
Community_ServicesGetResponseItems stored procedure	238	ContentSkin, basic skin type	79, 81–84
Community_Users table	168	control nomenclature, guidelines for	212
Community-Authenticated role	27	Controls subfolder, ISPAdmin folder	133
used in new website creation	120	ControlSkin, basic skin type	84
Community-Everyone role	27	copying files to server	
CommunityGlobals class	207	FrontPage access method	248
CommunityImageButton control	93	staging directory	249
CommunityInfo object	179	Xcopy deployment	248
community-specific Web Boxes, building	229–35	creating ascx files with Solution Explorer	218
Community_GetContentItem function	230	creating new modules	193
Page_Load method for current information display	232	Community_Faqs table	194, 197
components class, module classes	175	ContentInfo class, extending to module-specific use	203
configuring ASP.NET for CSK deployment	250–54, 250	design considerations	194
configuring the CSK project	126, 258	helper components	203
ISP Admin tools, configuring through	258	HTML markup, limitations of	194
Web.config configuration	258	step-by-step details	196
content class, module classes	179	stored procedure for FAQ table	198–202
content classes	208	stored procedures for FAQ table	198
content foundation, CSK module architecture		styles for mudules	215
see module architecture	165	user controls compared	217
ContentEditPage class	191, 210	Creating new modules	
ContentInfo class	175	design considerations	193–96
extending to module-specific use	203	current information, display of	232–34
ContentInfoObject	188	customization in CSK, content classes	208
ContentItemPage class	186	D	
		database deployment	254–57
		backup and restore	257
		database ownership issues	257
		DTS deployment	254
		script deployment	256
		DateCreated control	95

delegates, invoking	186	Edit Topics menu, Community Administration Menu	58
Delete Content menu, Community Administration Menu	66	Edit Users menu, Community Administration Menu	59
DELETE statement, deleting CSK content	174	Edit Voting Polls menu, Community Administration Menu	51
deploying CSK		Edit Web Box Services menu, Community Administration Menu	60
configuring ASP.NET	250	EditArticle method	176, 191
Copy Project command	248	EditCommunity subfolder, ISPAdmin folder	133
copying files to server	247–50	editing an existing FAQ, stored procedure for	199
CSK configured as a web application	252	editing messages	56
database deployment	254	editing topics	58
other websites, working with	259	enabling web services timer, Web.Config file	127
site configuration	251	Engine folder, basic CSK folder	134
design considerations for new modules		Enterprise Manager, SQL Server	254
displaying FAQ data, WebControl class	196	EventComparer	232
editing FAQ data	195	executionTimeout attribute, httpRuntime element	25
FAQ listings	194	F	
design considerations for websites	110	FaqAnswer control	206
DiscussEditContent control	90	FaqEditContent control	208
displaying list of elements, ContentList control	82	FaqReference control	206
DisplayQuota control	96	FaqSection class	209
E		FaqUtility class	204
Edit HTML link, using	116	FaqUtility.GetFaq method	209
Edit Messages menu, Community Administration Menu	56	Find in Files feature	88
Edit Sections menu, Community Administration Menu	39	FooterTemplate, template file	86
Edit Sections, Community Administration Menu	113	Framework subfolder, Engine folder	134
display options	42	FTP transfer of files to server	249
exchanging data with other communities	45		
functionality	39		
user level security	44		
Web Boxes	43		
Edit Service Subscriptions, Community Administration Menu	53		

G		installing CSK	3
GetArticleInfo method	177	multiple communities, support for	7
GetContentItemsDelegate delegate	209	ISPAdmin folder, basic CSK folder	132
GetControl method	210	ISPAdmin section, Web.Config file	127
GetFAQInfo method, FaqUtility class	205	ItemAuthor control	98
GetFaqs method, FaqUtility class	205	ItemBriefDescription control	92, 99
GetOptionalControl method	210	ItemDateCreated control	99
GetResponseInfo method, RSS feed	240	ItemDateList control	95
H		ItemEditContent control	99
HeaderTemplate, template file	86	ItemHasRead control	99
HTML Editor		ItemReadMoreLink control	100
support within CSK	32	ItemTemplate, template file	86
HTML View versus Design View	220	ItemTitle control	101
HtmlForm object	137	ItemTitleLink control	101
HtmlTextBox control	98	ItemViewCount control	102
HttpContext Items collection	182	L	
HttpContext.Cache object	134	LengthValidator control	102
HttpModule	238	LoadControl function	137
configuration of	239	loading the page skin	180
httpRuntime element	25	LoadSkin method	184
I		Logo control	103
image file handling, Web.Config files	128	Lunar .NET	2
ImageButton and CommunityImageButton controls compared	93	Lunar .NET community	11
ImageHandler class	65	main page	12
ImageItem control	100	modules available	15
Images subfolder, ISPAdmin folder	133	parent and child sections	25
ImageUtility class	65	M	
inheriting skins and styles	76	maintenance stored procedure	202
InitializeSkin method	186	named pages, registering of	202
inserting and updating module content, content classes	209	maxRequestLength attribute, httpRuntime element	25
		method signature, writing data access methods	209
			271

moderating websites	49	ordering content	169
module architecture, CSK		P	
content display broken into controls	206	Page_Init method	135
deleting content	174	PageFooter control	103
module classes	174	used in new website creation	120
retrieving articles	169, 173	PageHeader control	103
schematic diagram	165	PageSkins, basic skin type	79–81
skins and styles	191	ParentSection module	120
stored procedures	168	Placeholder controls	81
table structure	166	Play, sample community	111
module classes, CSK module architecture	174	populating community web service response objects	238
componetnts class	175	populating Web Boxex remotely	45
content class	179	previewing controls	209–12
Modules subfolder, Engine folder	134	processing web page requests	134
N		programmatic interaction with controls	210
named pages, registering of	202	Q	
namespace structure for storing components	203	Query Analyzer, executing script	257
new community, creating	70	quota errors, displaying	98
About Me section, adding	115	Quote.ascx file	67
ading forums	119	R	
Disign View, using for updates to styles	120	Really Simple Syndication (RSS)	43, 237
folder structure and skins, creating	111	Recent Logins, example control	
home section, adding functionality to	113	ascx file location	217, 218
populating the website	113	Solution Explorer, VS.NET, for creating ascx files	218
updating styles	120	stored procedure	218
news aggregator for RSS testing	244	recursive transfer of multiple files	249
notifications by e-mail	49	retrieving articles	173
O		retrieving content for a single FAQ, stored procedure for	201
objBuilder object	137	returning FAQs, stored procedure for	200
objPagePart object	136	RSS feed	
objPageSkin object	136		
objSectionInfo object	136		
obtaining references to controls	210		

advertising of	245	built-in with CSK	76
CommunityRSS.aspx file	240	composition	79
CreateRss method	240	new skins, developing	79
CreateRssTag method	240	predefined styles	78
GetResponseInfo method	240	Solution Explorer, VS.NET	
implementing	238	building themes	77
rss element	238	creating ascx files with	218
testing	244	Sorter control	105
S		SqlDataReader parameter	175
SampleData subfolder, ISPAdmin folder	133	stored procedures for table access	168
Scripts virtual directory	254	string transformations	43
SearchUtility class	177	styles	
SectionInfo object	134	built-in with CSK	76
SectionTitle control	104	subscribing to the website	53
used in new website creation	120	system roles in CSK	27
SectionTitle control for advertising RSS feed	245	T	
Send Newsletter menu, Community Administration Menu	62	table structure in CSK	166
sending newsletters	62	TemplateSkin, basic skin type	85
SeparatorTemplate, template file	86	themes	
ServiceLink property, SectionInfo object	241	personalized themes, building	75
ServiceResponseInfo class, community web service	238	Title control	106
ServiceResponseInfoItem class, community web service	238	try-catch-finally statement for closing database connections	205
setting themes	43	U	
site configuration	251	updating pages with stored procedures	
hosting in default website of Windows 2000	254	transactionless nature	200
hosting with a hosting provider	254	UploadValidator control	106
skin files	38, 75	user controls	217
SkinnedCommunityControl class	182, 195, 210	community-specific Web Box, building	229
skins		custom modules, implementing as user controls	223
		custom Web Boxes, building	224
		drawbacks	217
		Recent Logins, example control	217
		using existing CSS	221
		273	

utility class, static nature of the methods of	204	Web Matrix	3
V		web service versus Web Box service	45
validator, used for RSS testing	245	web services	45
VaryByParam versus VaryByControl, Web Boxes display	225	exposing data	46
View Activity Log menu, Community Administration Menu	65	RSS feed implementation, use for	238
View Documentation menu, Community Administration Menu	66	Web.Config file	126–29
ViewCount control	107	<compilation> element	128
viewing activity logs	65	<httpModules> element	128
viewing documentation	66	<webServices> element	129
W		basic sections	127
web applications, running under CSK	259	database connections section	127
Web Box service	45	Global.asax, code-behind	129
Web Boxes	43	ImageHandler class	128
community-specific Web Boxes	224	user information section	127
custom Web Boxes	223	web page requests handling	127
global Web Boxes	223	web services requests handling	127
populating remotely	45	WebBoxes	86
VaryByParam versus VaryByControl, effect on display	225	WebControl class	182, 196
		wrapper methods to invoke stored procedures	225
		X	
		Xcopy deployment	248

